AFRL-IF-RS-TR-2006-276
Final Technical Report
September 2006

KNOWLEDGE-INTENSIVE, INTERACTIVE AND EFFICIENT RELATIONAL PATTERN LEARNING

UNIVERSITY OF WISCONSIN-MADISON

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. L835/50

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-276 has been reviewed and is approved for publication.

APPROVED: /s/

CRAIG S. ANKEN
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA
Chief, Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
SEP 06	Final	Sep 01 – Mar 06

4. TITLE AND SUBTITLE
KNOWLEDGE-INTENSIVE, INTERACTIVE AND EFFICIENT RELATIONAL PATTERN LEARNING

5a. CONTRACT NUMBER
F30602-01-2-0571

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
31011G

6. AUTHOR(S)
David Page, Jude Shavlik

5d. PROJECT NUMBER
EELD

5e. TASK NUMBER
01

5f. WORK UNIT NUMBER
06

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Wisconsin-Madison
6743 Medical Sciences Center, 1300 University Ave.
Madison Wisconsin 53706

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFED
525 Brooks Road
Rome New York 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-276

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA #06-607

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes the work to develop and evaluate state-of-the-art relational pattern learning algorithms for the Evidence Assessment, Grouping, Linking and Evaluation (EAGLE) program. The University of Wisconsin team consisted of leaders in relational data mining and relational machine learning, in particular inductive logic programming (ILP). Major contributions by the team included the development of an ILP system implemented entirely in database operations, FOIL-D, and a statistical relational learning (SRL) system that incorporates explicit probabilistic constraints into ILP, CLP(BN). CLP(BN) incorporates all the representational power of probabilistic relational models (PRMs) but uses an ILP approach to learning. Another major contribution is the definition and development of View Learning, an approach to change of representation for SRL. Though SRL systems are particularly well-suited to the goals of EAGLE – indeed, the field of SRL received much of its impetus for growth from EAGLE – these systems have been constrained to work with the input representation, typically a relational schema. View Learning in SRL permits the definition of a new schema – a new view of the database – better suited to the learning goals. This project also made advances within learning ensembles, including the DECORATE approach to diverse ensembles, the use of bagging within ILP, the GLEANER algorithm to construct ensembles of relational rules having varying trade-offs of precision and recall, and a parallel implementation of bagging in ILP. The project also contributed novel stochastic search algorithms for ILP.

15. SUBJECT TERMS
Machine learning, inductive logic programming, probabilistic, data mining

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Craig S. Anken
U	U	U	UL	218	19b. TELEPONE NUMBER (*Include area code*)

Abstract

This report describes the work to develop and evaluate state-of-the-art relational pattern learning algorithms for the Evidence Assessment, Grouping, Linking and Evaluation (EAGLE) program. The University of Wisconsin team consisted of leaders in relational data mining and relational machine learning, in particular inductive logic programming (ILP). Major contributions by the team included the development of an ILP system implemented entirely in database operations, FOIL-D, and a statistical relational learning (SRL) system that incorporates explicit probabilistic constraints into ILP, CLP(BN). CLP(BN) incorporates all the representational power of probabilistic relational models (PRMs) but uses an ILP approach to learning. Another major contribution is the definition and development of View Learning, an approach to change of representation for SRL. Though SRL systems are particularly well-suited to the goals of EAGLE – indeed, the field of SRL received much of its impetus for growth from EAGLE – these systems have been constrained to work with the input representation, typically a relational schema. View Learning in SRL permits the definition of a new schema – a new view of the database – better suited to the learning goals. This project also made advances within learning ensembles, including the DECORATE approach to diverse ensembles, the use of bagging within ILP, the GLEANER algorithm to construct ensembles of relational rules having varying trade-offs of precision and recall, and a parallel implementation of bagging in ILP. The project also contributed novel stochastic search algorithms for ILP.

Each of the preceding contributions resulted in one or more publications in major computer science venues, such as the *International Joint Conference on Artificial Intelligence (IJCAI)*, the *International Conference on Uncertainty in Artificial Intelligence (UAI)*, the *International Conference on Machine Learning(ICML)*, the *International Conference on Inductive Logic Programming (ILP)*, the *International Conference on Intelligence Analysis (ICIA)*, the *National Conference on Artificial Intelligence (AAAI)*, and the *Machine Learning Journal*, and included the ICIA'05 Best Technical Paper and the ILP'04 Best Student Paper.

Table of Contents

Section 1: Executive Summary

Most algorithms for statistical classification, supervised machine learning or data mining assume every example or data point is represented by a feature-vector; a feature-vector specifies a value for each of a fixed set of variables or attributes. This feature-vector representation is assumed by popular and well-know learning algorithms such as support vector machines (SVMs), Bayesian network learning algorithms, decision tree learning algorithms, and ensembles of classifiers, including those constructed by boosting or bagging. On the other hand, detecting or predicting a threat event requires the analysis of data consisting of many objects – people, location, materials – and the relations among these objects – communications, purchases, transportation. Such data is inherently *relational* and cannot (at least as of this writing) be represented in feature-vector format without loss of information, change in statistical properties, or explosion in data size. Rather, such data is naturally represented in a relational database with multiple tables. Consequently, machine learning algorithms are needed that can directly analyze such relational data.

Inductive logic programming (ILP) has for over a decade been a leading approach to learning from relational data. But ILP at the start of this project had major shortcomings for analyzing threat events. These included long processing times and failure to incorporate and reason about probabilities. Our specific goals at the start of the program were to (1) incorporate explicit probabilities into ILP in an elegant manner (measurable objective: publication in a leading conference on probabilistic approaches), (2) reduce ILP processing times without reducing accuracy (measurable objective: cut processing time in half, not counting gains made because of the development of faster hardware), and (3) improve active learning methods (measurable objective: reduce training examples required for a given degree of accuracy). All three goals were met – see Sections 2.1, 2.3 and 2.4, respectively. A fourth goal was added, to implement a leading ILP algorithm completely in database operations; this goal was met with the system FOIL-D, now available in both MySQL and Oracle implementations.

Overall, the accomplishments of our project have substantially increased processing speed for ILP algorithms and have incorporated explicit probabilities in a natural way, *yielding leading approaches within the new area of statistical relational learning (SRL).* The resulting algorithms have been applied to all of the EAGLE evaluation databases; the application to prediction of aliases yielded the Best Technical Paper in the 2005 *International Conference on Intelligence Analysis.* In addition, these algorithms have been applied successfully to other relational databases of importance to society, including the prediction of malignancies among abnormalities on mammograms, the prediction of protein function from biological data, and automated information extraction from text.

Section 2: Technical Summary

This section presents the technical accomplishments of our project. Each accomplishment substantially extends capabilities in the state-of-the-art of relational learning. Each accomplishment has been tested on the EAGLE databases and/or other large real-world relational databases. For evaluation of our accomplishments, our group participated in every TIE except the first (which intentionally omitted pattern learning components). Our work on the last TIE led to the *2005 International Conference on Intelligence Analysis* Best Technical Paper, already mentioned. In addition to the TIEs, we also participated in the Cyc/21st Century Mini-TIE evaluating performance on real-world data (al Qaida). We focused on the tasks of predicting when two people were acquainted, and more generally when two people were "linked" (friends, acquaintances, relatives). Each accomplishment described below also is discussed in full detail in one or more associated, cited papers.

2.1. CLP(BN): Incorporating Explicit Probabilities into ILP

Datalog provides a standard theoretical underpinning for relational databases. Non-recursive Datalog with negation-as-failure is a restriction of first-order logic that is equivalent to relational algebra. In Datalog, missing values are represented by Skolem constants. More generally, in first-order logic missing values, or existentially-quantified variables, are represented by terms built from Skolem functors. In an analogy to probabilistic relational models (PRMs), we wish to represent the joint probability distribution over missing values in a database or logic program using a Bayesian network. We provide an extension of logic programs that makes it possible to specify a joint probability distribution over terms built from Skolem functors in the program. Our extension is based on constraint logic programming (CLP), where the constraints are Bayesian networks, so we call the extended language CLP(BN). We show that CLP(BN) subsumes PRMs; because CLP(BN) programs are logic programs, though, they can be learned by ILP algorithms with very simple modifications. CLP(BN) is publicly available as part of YAP Prolog at http://www.cos.ufrj.br/~vitor/Yap/clpbn. Details about CLP(BN) are available in the proceedings of UAI'03 [Santos Costa *et al*., 2003].

2.2. View Learning for Statistical Relational Learning

Statistical relational learning (SRL) constructs probabilistic models from relational databases. A key capability of SRL is the learning of arcs (in the Bayes net sense) connecting entries in different rows of a relational table, or in different tables. Nevertheless, SRL approaches as of 2004 were constrained to use the existing database schema. For many database applications, users find it profitable to define alternative "views" of the database, in effect defining new fields or tables. Such new fields or tables can also be highly useful in learning. We provided SRL with the capability of learning such new views. The view learning approach is based on ILP to learn the definitions of new fields in a database. View learning was shown to significantly improve performance of SRL on the task of predicting whether abnormalities in a mammogram are malignant

or benign. Details about our view learning approach are available in an IJCAI'05 paper [Davis *et al*., 2005].

2.3. Stochastic Search in Inductive Logic Programming

Recent statistical performance studies of search algorithms in difficult combinatorial problems have demonstrated the benefits of randomising and restarting the search procedure. Specifically, it has been found that if the search cost distribution of the non-restarted randomised search exhibits a slower-than-exponential decay (that is, a "heavy tail"), restarts can reduce the search cost expectation. We report on an empirical study of randomized restarted search in ILP. Our experiments conducted on a high-performance distributed computing platform provide an extensive statistical performance sample of five search algorithms operating on two principally different classes of ILP problems, one represented by an artificially generated graph problem and the other by three traditional classification benchmarks (mutagenicity, carcinogenicity, finite element mesh design). The sample allows us to (1) estimate the conditional expected value of the search cost (measured by the total number of clauses explored) given the minimum clause score required and a "cutoff" value (the number of clauses examined before the search is restarted), (2) estimate the conditional expected clause score given the cutoff value and the invested search cost, and (3) compare the performance of randomised restarted search strategies to a deterministic non-restarted search. Our findings indicate striking similarities across the five search algorithms and the four domains, in terms of the basic trends of both the statistics (1) and (2). Also, we observe that the cutoff value is critical for the performance of the search algorithm, and using its optimal value in a randomised restarted search may decrease the mean search cost (by several orders of magnitude) or increase the mean achieved score significantly with respect to that obtained with a deterministic non-restarted search. Further details are available in a *Machine Learning Journal* paper as well as two ILP papers [Zelezny et al., 2003, 2004, 2006].

2.4. The DECORATE Algorithm for Enhancing Diversity in Ensembles and for Active Learning

We developed a new meta-learner, **DECORATE** (Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples), for building diverse ensembles of classifiers by using specially constructed artificial training examples [Melville et al., 2003]. Experiments demonstrate that our method performs consistently better than bagging and Random Forests. It also obtains higher accuracy than boosting on small training sets, and achieves comparable performance on larger training sets. We also demonstrated the resilience of DECORATE to three types of imperfections in data: missing features, classification noise and feature noise. DECORATE is now publicly available as part of the standard WEKA data mining software package.

Based on DECORATE, we developed a new method for *active learning* to significantly reduce the amount of supervised data required for effective learning. Our approach uses DECORATE ensembles to select the most useful examples to be labeled for training. Experimental results show that our approach produces accurate classifiers with fewer training examples than other Query by Committee approaches to active learning. In applications such as fraud detection, credit ranking, and direct marketing it is

also critical to have good class probability estimates. For such applications, we have shown that Jensen-Shannon divergence (a similarity measure for probability distributions) can be used to improve active learning for probability estimation [Melville *et al.,* 2004].

2.5. ILP in Database Operations: FOIL-D

Multi-relational rule mining is important for knowledge discovery in relational databases as it allows for discovery of patterns involving multiple relational tables. ILP techniques have had considerable success on a variety of multi-relational rule mining tasks. Nevertheless, because ILP implementations operate in RAM, most ILP systems do not scale to very large relational databases. We performed two major extensions to one of the most popular ILP systems, FOIL, to improve its scalability. First, we showed how to implement FOIL entirely in database operations, rather than in RAM, and constructed both Oracle and MySQL implementations. The resulting system we called FOIL in Databases, or FOIL-D. FOIL-D enables the FOIL algorithm to run on data sets that previously had been out of its scope. Second, we developed estimation methods, based on histograms as used widely in database management systems, that significantly decrease the computational cost of learning a set of rules. The full paper [Bockhorst & Ong, 2004] presents detailed experimental results that indicate that on a set of standard ILP datasets, the rule sets learned using our extensions are equivalent to those learned with standard FOIL but at considerably less cost.

2.6. Varying Precision-Recall Tradeoff Among ILP-Induced Rules: GLEANER

Many relational domains, including those in EAGLE, involve highly unbalanced data. Most people or actions in a database are not threats; most names are not aliases; most possible attacks are never made. Another task involving unbalanced data is Information Extraction (IE), a task that typically involves many more negative examples than positive examples. IE is the process of finding facts in unstructured text, such as biomedical journals, and putting those facts in an organized system. In particular, we have focused on learning to recognize instances of the protein-localization relationship in Medline abstracts. A common way to measure performance in these domains is to use precision and recall instead of simply using accuracy. We developed Gleaner, a randomized search method which collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an “at least N of these M clauses" thresholding method to combine the selected clauses. We compared Gleaner to ensembles of standard Aleph theories on the IE task and observed that Gleaner produces comparable testset results in a fraction of the training time needed for ensembles [Goadrich *et al.,* 2004]. We also applied Gleaner to the challenge task for the Learning Language in Logic Workshop with success [Goadrich *et al.*, 2005].

Section 3: List of Publications

The details of the technical accomplishments discussed in the previous section appear in the following publications in major computer science venues. All these venues are rigorously refereed, and the papers gratefully acknowledge the EAGLE/Air Force grant F30602-01-2-0571.

1. P. Melville and R. Mooney (2003). Constructing Diverse Classifier Ensembles Using Artificial Training Examples. Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), pp. 505-510.
2. L. Tang, R. Mooney and P. Melville (2003). Scaling Up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism. KDD-03 Workshop on Multi-Relational Data Mining, Washington DC.
3. Santos Costa, V., Page, D., Qazi, M. and Cussens, J. CLP(BN): Constraint Logic Programming for Probabilistic Knowledge. *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence* (UAI'03). U. Kjaerulff and C. Meek (Ed.s) San Francisco: Morgan Kaufmann Publishers, pp. 517-524, 2003.
4. Dutra, I., Page, D., Santos Costa, V. and Shavlik, J. An Empirical Evaluation of Bagging in Inductive Logic Programming. *Proceedings of the Twelfth International Conference on Inductive Logic Programming* (ILP'02), published as Lecture Notes in Computer Science 2583, Springer 2003, S. Matwin and C. Sammut (Ed.s), pp. 48-65, 2003.
5. Zelezny, F., Srinivasan, A. and Page, D. Lattice-Search Runtime Distributions May be Heavy-Tailed. Proceedings *of the Twelfth International Conference on Inductive Logic Programming* (ILP'02), published as Lecture Notes in Computer Science 2583, Springer 2003, S. Matwin and C. Sammut (Ed.s), pp. 333-345, 2003.
6. Mooney, R., Melville, P., Tang, L., Shavlik, J., Dutra, I. and Page, D. Relational Data Mining with Inductive Logic Programming for Link Discovery. In H. Kargupta and A. Joshi (Ed.s), *Data Mining: Next Generation Challenges and Future Directions*. Cambridge, MA: MIT/AAAI Press. 2003.
7. Page, D. and Srinivasan, A. ILP: A Short Look Back and a Longer Look Forward. *Journal of Machine Learning Research*, 4, pp. 415-430, 2003.
8. J. Bockhorst and I. Ong. FOIL-D: Efficiently Scaling FOIL for Multi-relational Data Mining of Large Datasets. Proceedings of the Fourteenth International Conference on Inductive Logic Programming, Porto, Portugal, 2004.

9. J. Davis, V. Santos Costa, I. Ong, D. Page, and I. Dutra. Using Bayesian Classifiers to Combine Rules. KDD Workshop on Multi-relational Data Mining, 2004.

10. F. DiMaio and J. Shavlik. Learning an Approximation to Inductive Logic Programming Clause Evaluation. Proceedings of the Fourteenth International Conference on Inductive Logic Programming, Porto, Portugal, 2004.

11. M. Goadrich, L. Oliphant and J. Shavlik. Learning Ensembles of First-Order Clauses for Recall-Precision Curves: A Case Study in Biomedical Information Extraction. Proceedings of the Fourteenth International Conference on Inductive Logic Programming, Porto, Portugal, 2004. **Winner of Best Student Paper Award. Longer version *to appear in* special issue of *Machine Learning Journal,* on ILP, 2006**.

12. F. Zelezny, A. Srinivasan and D. Page. A Monte Carlo Study of Randomized Restarted Search**. Longer version of ILP'04 paper, *To appear in* special issue of *Machine Learning Journal,* on ILP, 2006**

13. J. Davis, I. Dutra, D. Page and V. Santos Costa. Establishing Identity Equivalence in Multi-Relational Domains. *To appear* in the *Proceedings of the International Conference on Intelligence Analysis (ICIA'05).* ***Winner of ICIA'05 Best Technical Paper Award.***

14. J. Davis, E. Burnside, I. Dutra, D. Page, R. Ramakrishnan, V. Santos Costa and J. Shavlik. View Learning for Statistical Relational Learning: With an Application to Mammography . *Proceedings of the International Joint Conference on Artificial Intelligence, 2005.*

15. H. Corrada Bravo, D. Page, R. Ramakrishnan, J. Shavlik and V. Santos Costa. A Framework for Set-Oriented Computation in Inductive Logic Programming and its Application in Generalizing Inverse Entailment. *Fifteenth International Conference on Inductive Logic Programming (ILP'05), 2005.*

16. M. Goadrich, L. Oliphant and J. Shavlik. Learning to Extract Genic Interactions using Gleaner. Proceedings of the Fourth International Workshop on Learning Language in Logic, Bonn, Germany, 2005.

Section 4: Appendix -- Published Papers

This section includes in their entirety each of our papers published under this project, as surveyed in the Technical Summary (Section 2) and listed in Section 3. These papers include the detailed Accuracies, Precision-Recall curves, ROC curves and run-times discussed in Section 2 as showing that we met and exceeded the quantitative objectives of our project. Some papers cover additional data sets on which we evaluated our algorithms as well, included the prediction of which abnormalities on a mammogram are malignant, the analysis of synthetic graph data, and information extraction from text.

Proceedings of the IJCAI-2003, pp.505-510,
Acapulco, Mexico, August 2003

Constructing Diverse Classifier Ensembles using Artificial Training Examples

Prem Melville and Raymond J. Mooney
Department of Computer Sciences
University of Texas
1 University Station, C0500
Austin, TX 78712
melville@cs.utexas.edu, mooney@cs.utexas.edu

Abstract

Ensemble methods like bagging and boosting that combine the decisions of multiple hypotheses are some of the strongest existing machine learning methods. The diversity of the members of an ensemble is known to be an important factor in determining its generalization error. This paper presents a new method for generating ensembles that directly constructs diverse hypotheses using additional artificially-constructed training examples. The technique is a simple, general meta-learner that can use any strong learner as a base classifier to build diverse committees. Experimental results using decision-tree induction as a base learner demonstrate that this approach consistently achieves higher predictive accuracy than both the base classifier and bagging (whereas boosting can occasionally decrease accuracy), and also obtains higher accuracy than boosting early in the learning curve when training data is limited.

1 Introduction

One of the major advances in inductive learning in the past decade was the development of *ensemble* or *committee* approaches that learn and retain multiple hypotheses and combine their decisions during classification [Dietterich, 2000]. For example, *boosting* [Freund and Schapire, 1996], an ensemble method that learns a series of "weak" classifiers each one focusing on correcting the errors made by the previous one, has been found to be one of the currently best generic inductive classification methods [Hastie *et al.*, 2001].

Constructing a *diverse* committee in which each hypothesis is as different as possible (decorrelated with other members of the ensemble) while still maintaining consistency with the training data is known to be a theoretically important property of a good committee [Krogh and Vedelsby, 1995]. Although all successful ensemble methods encourage diversity to some extent, few have focused directly on the goal of maximizing diversity. Existing methods that focus on achieving diversity [Opitz and Shavlik, 1996; Rosen, 1996] are fairly complex and are not general *meta-learners* like bagging [Breiman, 1996] and boosting that can be applied to any base learner to produce an effective committee [Witten and Frank, 1999].

We present a new meta-learner (DECORATE, Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples) that uses an existing "strong" learner (one that provides high accuracy on the training data) to build an effective diverse committee in a fairly simple, straightforward manner. This is accomplished by adding different randomly constructed examples to the training set when building new committee members. These artificially constructed examples are given category labels that *disagree* with the current decision of the committee, thereby easily and directly increasing diversity when a new classifier is trained on the augmented data and added to the committee.

Boosting and bagging provide diversity by sub-sampling or re-weighting the existing training examples. If the training set is small, this limits the amount of ensemble diversity that these methods can obtain. DECORATE ensures diversity on an arbitrarily large set of additional artificial examples. Therefore, one hypothesis is that it will result in higher generalization accuracy when the training set is small. This paper presents experimental results on a wide range of UCI data sets comparing boosting, bagging, and DECORATE, all using J48 decision-tree induction (a Java implementation of C4.5 [Quinlan, 1993] introduced in [Witten and Frank, 1999]) as a base learner. Cross-validated learning curves support the hypothesis that "DECORATEd trees" generally result in greater classification accuracy for small training sets.

2 Ensembles and Diversity

In an ensemble, the combination of the output of several classifiers is only useful if they disagree on some inputs [Krogh and Vedelsby, 1995]. We refer to the measure of disagreement as the $diversity$ of the ensemble. There have been several methods proposed to measure ensemble diversity [Kuncheva and Whitaker, 2002] — usually dependent on the measure of accuracy. For regression, where the mean squared error is commonly used to measure accuracy, variance can be used as a measure of diversity. So the diversity of the i^{th} classifier on example x can be defined as $d_i(x) = [C_i(x) - C^*(x)]^2$, where $C_i(x)$ and $C^*(x)$ are the predictions of the i^{th} classifier and the ensemble respectively. For this setting Krogh et al [1995] show that the generalization error, E, of the ensemble can be expressed as $E = \bar{E} - \bar{D}$, where $\bar{E}$ and $\bar{D}$ are the mean error and diversity of the ensemble respectively.

For classification problems, where the 0/1 loss function is most commonly used to measure accuracy, the diversity of the i^{th} classifier can be defined as:

$$d_i(x) = \begin{cases} 0: \text{ if } C_i(x) = C^*(x) \\ 1: \text{otherwise} \end{cases} \quad (1)$$

However, in this case the above simple linear relationship does not hold between E, $\bar{E}$ and $\bar{D}$. But there is still strong reason to believe that increasing diversity should decrease ensemble error [Zenobi and Cunningham, 2001]. The underlying principle of our approach is to build ensembles of classifiers that are consistent with the training data and maximize diversity as defined in (1).

3 DECORATE: Algorithm Definition

In DECORATE (see Algorithm 1), an ensemble is generated iteratively, learning a classifier at each iteration and adding it to the current ensemble. We initialize the ensemble to contain the classifier trained on the given training data. The classifiers in each successive iteration are trained on the original training data and also on some artificial data. In each iteration artificial training examples are generated from the data distribution; where the number of examples to be generated is specified as a fraction, R_{size}, of the training set size. The labels for these artificially generated training examples are chosen so as to differ maximally from the current ensemble's predictions. The construction of the artificial data is explained in greater detail in the following section. We refer to the labeled artificially generated training set as the *diversity data*. We train a new classifier on the union of the original training data and the diversity data. If adding this new classifier to the current ensemble increases the ensemble training error, then we reject this classifier, else we add it to the current ensemble. This process is repeated until we reach the desired committee size or exceed the maximum number of iterations.

To classify an unlabeled example, x, we employ the following method. Each base classifier, C_i, in the ensemble C^* provides probabilities for the class membership of x. If $P_{C_i,y}(x)$ is the probability of example x belonging to class y according to the classifier C_i, then we compute the class membership probabilities for the entire ensemble as:

$$P_y(x) = \frac{\sum_{C_i \in C^*} P_{C_i,y}(x)}{|C^*|}$$

where $P_y(x)$ is the probability of x belonging to class y. We then select the most probable class as the label for x i.e. $C^*(x) = argmax_{y \in Y}\, P_y(x)$

3.1 Construction of Artificial Data

We generate artificial training data by randomly picking data points from an approximation of the training-data distribution. For a numeric attribute, we compute the mean and standard deviation from the training set and generate values from the Gaussian distribution defined by these. For a nominal attribute, we compute the probability of occurrence of each distinct value in its domain and generate values based on this distribution. We use Laplace smoothing so that nominal attribute values not represented in the training set still have a non-zero probability of occurrence. In constructing artificial data points, we make the simplifying assumption that the attributes are independent. It is possible to more accurately estimate the joint probability distribution of the attributes; but this would be time consuming and require a lot of data.

In each iteration, the artificially generated examples are labeled based on the current ensemble. Given an example, we first find the class membership probabilities predicted by the ensemble, replacing zero probabilities with a small non-zero value. Labels are then selected, such that the probability of selection is inversely proportional to the current ensemble's predictions.

Algorithm 1 The DECORATE algorithm

Input:
$BaseLearn$ - base learning algorithm
T - set of m training examples $< (x_1, y_1), ..., (x_m, y_m) >$ with labels $y_j \in Y$
C_{size} - desired ensemble size
I_{max} - maximum number of iterations to build an ensemble
R_{size} - factor that determines number of artificial examples to generate

1. $i = 1$
2. $trials = 1$
3. $C_i = BaseLearn(T)$
4. Initialize ensemble, $C^* = \{C_i\}$
5. Compute ensemble error, $\epsilon = \frac{\sum_{x_j \in T: C^*(x_j) \neq y_j} 1}{m}$
6. While $i < C_{size}$ and $trials < I_{max}$
7. Generate $R_{size} \times |T|$ training examples, R, based on distribution of training data
8. Label examples in R with probability of class labels inversely proportional to C^*'s predictions
9. $T = T \bigcup R$
10. $C' = BaseLearn(T)$
11. $C^* = C^* \bigcup \{C'\}$
12. $T = T - R$, remove the artificial data
13. Compute training error, ϵ', of C^* as in step 5
14. If $\epsilon' \leq \epsilon$
15. $i = i + 1$
16. $\epsilon = \epsilon'$
17. otherwise,
18. $C^* = C^* - \{C'\}$
19. $trials = trials + 1$

4 Experimental Evaluation

4.1 Methodology

To evaluate the performance of DECORATE we ran experiments on 15 representative data sets from the UCI repository [Blake and Merz, 1998] used in similar studies [Webb, 2000;

Quinlan, 1996]. We compared the performance of DECORATE to that of Adaboost, Bagging and J48, using J48 as the base learner for the ensemble methods and using the Weka implementations of these methods [Witten and Frank, 1999]. For the ensemble methods, we set the ensemble size to 15. Note that in the case of DECORATE, we only specify a maximum ensemble size, the algorithm terminates if the number of iterations exceeds the maximum limit even if the desired ensemble size is not reached. For our experiments, we set the maximum number of iterations in DECORATE to 50. We ran experiments varying the amount of artificially generated data, R_{size}; and found that the results do not vary much for the range 0.5 to 1. However, R_{size} values lower than 0.5 do adversely affect DECORATE, because there is insufficient artificial data to give rise to high diversity. The results we report are for R_{size} set to 1, i.e. the number of artificially generated examples is equal to the training set size.

The performance of each learning algorithm was evaluated using 10 complete 10-fold cross-validations. In each 10-fold cross-validation each data set is randomly split into 10 equal-size segments and results are averaged over 10 trials. For each trial, one segment is set aside for testing, while the remaining data is available for training. To test performance on varying amounts of training data, learning curves were generated by testing the system after training on increasing subsets of the overall training data. Since we would like to summarize results over several data sets of different sizes, we select different *percentages* of the total training-set size as the points on the learning curve.

To compare two learning algorithms across all domains we employ the statistics used in [Webb, 2000], namely the win/draw/loss record and the geometric mean error ratio. The win/draw/loss record presents three values, the number of data sets for which algorithm A obtained better, equal, or worse performance than algorithm B with respect to classification accuracy. We also report the *statistically significant* win/draw/loss record; where a win or loss is only counted if the difference in values is determined to be significant at the 0.05 level by a paired t-test. The geometric mean error ratio is defined as $\sqrt[n]{\prod_{i=1}^{n} \frac{E_A}{E_B}}$, where E_A and E_B are the mean errors of algorithm A and B on the same domain. If the geometric mean error ratio is less than one it implies that algorithm A performs better than B, and vice versa. We compute error ratios so as to capture the degree to which algorithms out-perform each other in win or loss outcomes.

4.2 Results

Our results are summarized in Tables 1-3. Each cell in the tables presents the accuracy of DECORATE versus another algorithm. If the difference is statistically significant, then the larger of the two is shown in bold. We varied the training set sizes from 1-100% of the total available data, with more points lower on the learning curve since this is where we expect to see the most difference between algorithms. The bottom of the tables provide summary statistics, as discussed above, for each of the points on the learning curve.

DECORATE has more *significant* wins to losses over Bagging for all points along the learning curve (see Table 2). DECORATE also outperforms Bagging on the geometric mean ratio. This suggests that even in cases where Bagging beats DECORATE the improvement is less than DECORATE's improvement on Bagging on the rest of the cases.

DECORATE outperforms AdaBoost early on the learning curve both on significant wins/draw/loss record and geometric mean ratio; however, the trend is reversed when given 75% or more of the data. Note that even with large amounts of training data, DECORATE's performance is quite competitive with Adaboost - given 100% of the data DECORATE produces higher accuracies on 6 out of 15 data sets.

It has been observed in previous studies [Webb, 2000; Bauer and Kohavi, 1999] that while AdaBoost usually significantly reduces the error of the base learner, it occasionally increases it, often to a large extent. DECORATE does not have this problem as is clear from Table 1.

On many data sets, DECORATE achieves the same or higher accuracy as Bagging and AdaBoost with many fewer training examples. Figure 1 show learning curves that clearly demonstrate this point. Hence, in domains where little data is available or acquiring labels is expensive, DECORATE has an advantage over other ensemble methods.

We performed additional experiments to analyze the role that diversity plays in error reduction. We ran DECORATE at 10 different settings of R_{size} ranging from 0.1 to 1.0, thus varying the diversity of ensembles produced. We then compared the diversity of ensembles with the reduction in generalization error. Diversity of an ensemble is computed as the mean diversity of the ensemble members (as given by Eq. 1). We compared ensemble diversity with the *ensemble error reduction*, i.e. the difference between the average error of the ensemble members and the error of the entire ensemble (as in [Cunningham and Carney, 2000]). We found that the correlation coefficient between diversity and ensemble error reduction is 0.6225 ($p^1 \ll 10^{-50}$), which is fairly strong. Furthermore, we compared diversity with the *base error reduction*, i.e. the difference between the error of the base classifier and the ensemble error. The base error reduction gives a better indication of the improvement in performance of an ensemble over the base classifier. The correlation of diversity versus the base error reduction is 0.1552 ($p \ll 10^{-50}$). We note that even though this correlation is weak, it is still a *statistically significant* positive correlation. These results reinforce our belief that increasing ensemble diversity is a good approach to reducing generalization error.

To determine how the performance of DECORATE changes with ensemble size, we ran experiments with increasing sizes. We compared results for training on 20% of available data, since the advantage of DECORATE is most noticeable low on the learning curve. Due to lack of space, we do not include the results for all 15 datasets, but present five representative datasets (see Figure 2). The performance on other datasets is similar. We note, in general, that the accuracy of DECORATE increases with ensemble size; though on most datasets, the performance levels out with an ensemble size of 10 to 25.

[1] The p-value is the probability of getting a correlation as large as the observed value by random chance, when the true correlation is zero.

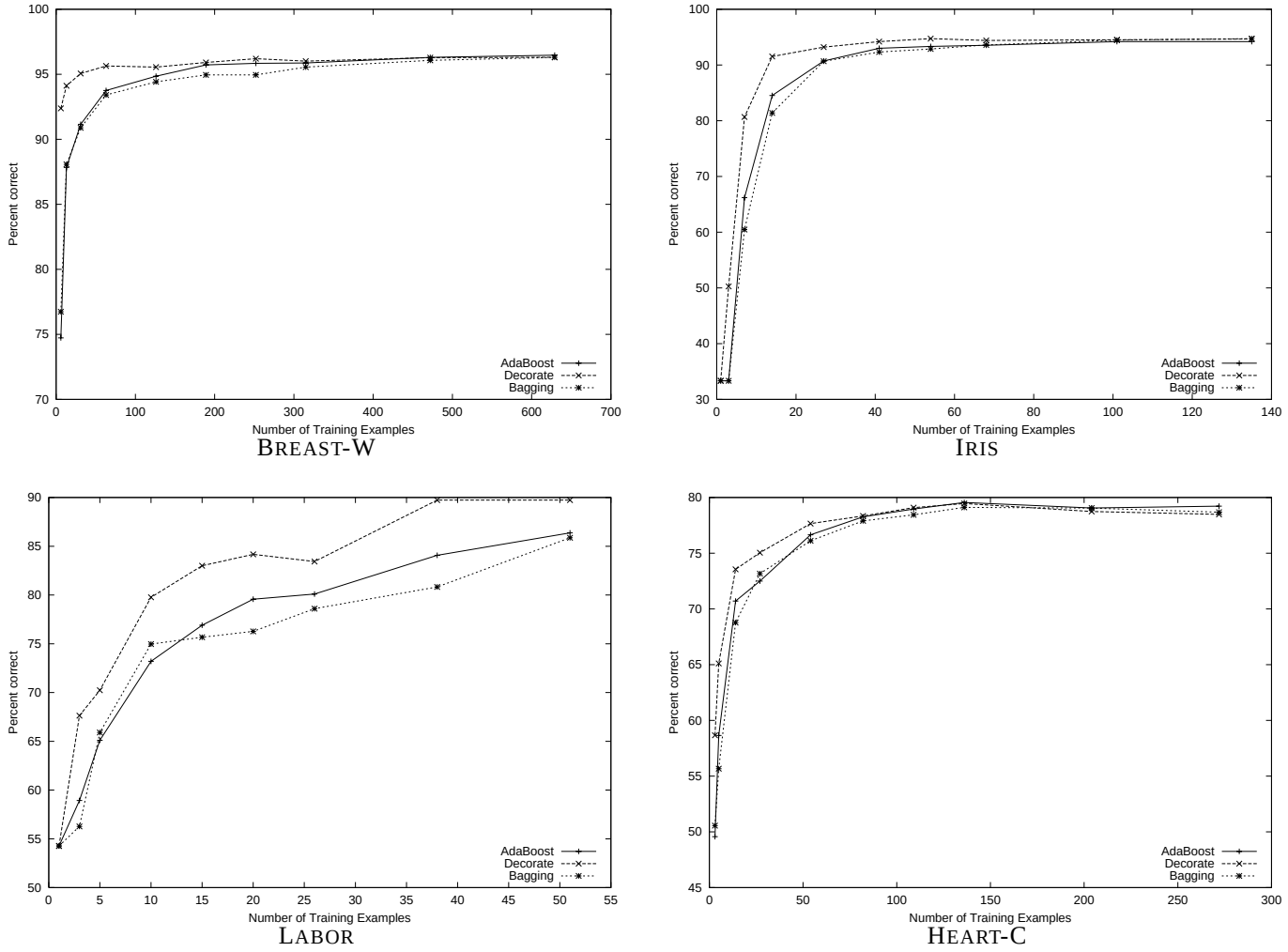


Figure 1: DECORATE compared to AdaBoost and Bagging

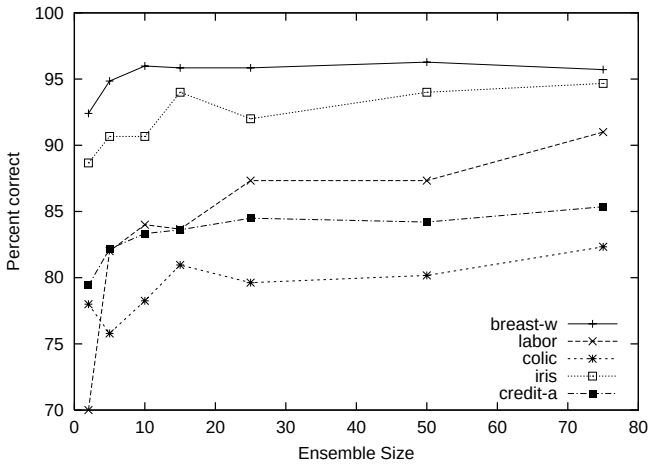


Figure 2: DECORATE at different ensemble sizes

5 Related Work

There have been some other attempts at building ensembles that focus on the issue of diversity. Liu et al [1999] and Rosen [1996] simultaneously train neural networks in an ensemble using a correlation penalty term in their error functions. Opitz and Shavlik [1996] use a genetic algorithm to search for a good ensemble of networks. To guide the search they use an objective function that incorporates both an accuracy and diversity term. Zenobi et al [2001] build ensembles based on different feature subsets; where feature selection is done using a hill-climbing strategy based on classifier error and diversity. A classifier is rejected if the improvement of one of the metrics lead to a "substantial" deterioration of the other; where "substantial" is defined by a pre-set threshold.

In all these approaches, ensembles are built attempting to simultaneously optimize the accuracy and diversity of individual ensemble members. However, in DECORATE, our goal is to minimize *ensemble error* by increasing diversity. At no point does the training accuracy of the ensemble go below

Table 1: DECORATE vs J48

Dataset	1%	2%	5%	10%	20%	30%	40%	50%	75%	100%
anneal	**75.29**/72.49	**78.14**/75.31	**85.24**/82.08	**92.26**/89.28	**96.48**/95.57	**97.36**/96.47	**97.73**/97.3	**98.16**/97.93	98.39/98.35	98.71/98.55
audio	16.66/16.66	23.73/23.07	41.72/41.17	**55.42**/51.67	**64.09**/60.59	**67.62**/64.84	**70.46**/68.11	**72.82**/70.77	**77.8**/75.15	**82.1**/77.22
autos	24.33/24.33	29.6/29.01	**36.73**/34.37	42.89/41.22	52.2/50.53	**59.86**/53.92	**64.77**/59.68	**68.6**/65.24	**78**/73.15	**83.64**/81.72
breast-w	**92.38**/74.73	**94.12**/87.34	**95.06**/89.42	**95.64**/92.21	**95.55**/93.09	**95.91**/93.36	**96.2**/93.85	**96.01**/94.24	**96.28**/94.65	**96.31**/95.01
credit-a	71.78/69.54	74.83/**77.46**	80.61/81.57	83.09/82.35	84.38/84.29	84.68/84.59	**85.22**/84.41	**85.57**/84.78	85.61/85.43	85.93/85.57
Glass	31.69/31.69	**35.86**/32.96	**44.5**/38.34	**55.4**/46.62	**61.77**/54.16	**66.01**/60.63	**68.07**/61.38	**68.85**/63.69	**72.73**/67.53	**72.77**/67.77
heart-c	**58.66**/49.57	**65.11**/58.03	**73.55**/67.71	**75.05**/70.15	**77.66**/73.44	**78.34**/74.61	**79.09**/74.78	**79.46**/75.62	**78.74**/76.7	**78.48**/77.17
hepatitis	52.33/52.33	**71.95**/65.93	**76.59**/72.75	78.85/78.25	80.28/78.61	**81.14**/78.63	**81.53**/79.35	**81.68**/79.57	**82.37**/79.04	**82.43**/79.22
colic	**59.85**/52.85	**68.19**/65.31	74.91/74.37	78.45/79.94	81.81/82.71	82.47/83.41	82.74/83.55	83.5/**84.66**	83.93/**85.18**	85.24/85.16
iris	33.33/33.33	**50.87**/33.33	**80.67**/59.33	**91.27**/84.33	**93.07**/91.33	**94.4**/92.73	**95.07**/93	94.07/93.33	94.67/94.07	94.93/94.73
labor	54.27/54.27	54.27/54.27	**67.7**/58.93	**71.47**/64.77	**78.6**/70.07	**81.67**/73.7	**85.67**/75.17	**84.2**/75.8	**87.53**/77.4	**89.5**/78.8
lymph	48.39/48.39	**53.49**/46.64	**65.73**/60.39	**72.79**/68.21	**74.57**/70.79	**78.84**/73.58	**78.37**/74.53	**78.31**/73.34	**78.06**/75.63	**78.74**/76.06
segment	**67.94**/52.43	**80.75**/73.26	**89.52**/85.41	**92.87**/89.34	**94.99**/92.22	**95.82**/93.37	**96.54**/94.34	**96.93**/94.77	**97.56**/95.94	**98.02**/96.79
soybean	**19.37**/13.69	**32.12**/22.32	**55.55**/42.94	**73.51**/59.04	**84.63**/74.49	**88.52**/81.59	**90.37**/84.78	**91.35**/86.89	**92.85**/89.44	**93.81**/91.76
splice	**63.48**/59.92	67.56/**68.69**	77.34/77.49	82.62/82.58	**88.2**/87.98	90.46/90.44	91.82/91.77	92.5/92.4	93.41/93.47	93.92/**94.03**
Win/Draw/Loss	15/0/0	13/0/2	13/0/2	14/0/1	14/0/1	14/0/1	14/0/1	14/0/1	13/0/2	14/0/1
Sig. W/D/L	7/8/0	10/3/2	11/4/0	10/5/0	11/4/0	12/3/0	13/2/0	12/2/1	10/4/1	10/4/1
GM error ratio	0.858	0.8649	0.8116	0.8098	0.8269	0.8103	0.7983	0.8305	0.8317	0.8293

Table 2: DECORATE vs Bagging

Dataset	1%	2%	5%	10%	20%	30%	40%	50%	75%	100%
anneal	75.29/74.57	**78.14**/76.42	**85.24**/82.88	**92.26**/89.87	**96.48**/95.67	**97.36**/96.89	**97.73**/97.34	**98.16**/97.78	98.39/98.53	98.71/98.83
audio	**16.66**/12.98	23.73/23.68	**41.72**/38.55	**55.42**/51.34	**64.09**/61.76	67.62/66.9	70.46/70.29	72.82/73.07	77.8/77.32	**82.1**/80.71
autos	**24.33**/22.16	29.6/28	36.73/35.88	42.89/44.65	52.2/54.32	59.86/59.67	64.77/65.6	68.6/69.88	78/77.97	83.64/83.12
breast-w	**92.38**/76.74	**94.12**/88.07	**95.06**/90.88	**95.64**/93.41	**95.55**/94.42	**95.91**/94.95	**96.2**/94.95	**96.01**/95.55	96.28/96.07	96.31/96.3
credit-a	71.78/69.54	74.83/**77.99**	80.61/**82.58**	83.09/83.9	84.38/**85.13**	84.68/**85.78**	85.22/85.59	85.57/85.64	85.61/**86.12**	85.93/85.96
Glass	**31.69**/24.85	**35.86**/31.47	**44.5**/40.87	**55.4**/49.6	**61.77**/58.9	**66.01**/64.35	68.07/66.3	68.85/68.44	72.73/72	72.77/**74.67**
heart-c	**58.66**/50.56	**65.11**/55.67	**73.55**/68.77	**75.05**/73.17	**77.66**/76.12	78.34/77.9	79.09/78.44	79.46/79.11	78.74/79.05	78.48/78.68
hepatitis	52.33/52.33	**72.14**/63.18	76.8/75.2	79.48/78.64	80.7/80.42	81.81/81.07	81.65/81.22	**83.19**/81.06	**82.99**/80.87	82.62/81.34
colic	**58.37**/53.14	**66.58**/63.83	75.85/76.44	79.54/80.06	81.33/**83.04**	82.47/**83.58**	83.02/**83.98**	83.1/**84.47**	84.02/**85.4**	84.69/85.34
iris	33.33/33.33	**50.27**/33.33	**80.67**/60.47	**91.53**/81.4	**93.2**/90.67	**94.2**/92.33	**94.73**/92.87	**94.4**/93.6	94.53/94.47	94.67/94.73
labor	54.27/54.27	54.27/54.27	**67.63**/56.27	**70.23**/65.9	**79.77**/74.97	**83**/75.67	**84.17**/76.27	**83.43**/78.6	**89.73**/80.83	**89.73**/85.87
lymph	48.39/48.39	**53.62**/47.11	**65.06**/60.12	71.2/69.68	**76.74**/73.6	**78.84**/76.58	78.17/77.68	**78.99**/76.98	**79.14**/76.8	79.08/77.97
segment	**67.03**/55.88	**81.16**/76.36	**89.61**/87.42	**92.83**/91.01	**94.88**/93.4	**95.94**/94.65	**96.47**/95.26	**96.93**/95.82	**97.58**/96.78	**98.03**/97.41
soybean	**19.51**/14.56	**32.4**/24.58	**55.36**/47.46	**73.06**/65.45	**85.14**/79.29	**88.27**/85.05	**90.22**/87.89	**91.4**/89.22	**92.75**/91.56	**93.89**/92.71
splice	62.77/62.52	67.8/**72.36**	77.37/**80.5**	82.55/**85.44**	88.24/**89.5**	90.47/**91.44**	91.84/**92.4**	92.41/**93.07**	93.44/**94.06**	93.92/**94.53**
Win/Draw/Loss	15/0/0	13/0/2	12/0/3	11/0/4	11/0/4	12/0/3	11/0/4	10/0/5	10/0/5	8/0/7
Sig. W/D/L	8/7/0	10/3/2	10/3/2	9/5/1	10/2/3	8/4/3	6/7/2	8/5/2	5/7/3	4/9/2
GM error ratio	0.8727	0.8785	0.8552	0.8655	0.8995	0.9036	0.8979	0.9214	0.9312	0.9570

that of the base classifier; however, this is a possibility with previous methods. Furthermore, none of the previous studies compared their methods with the standard ensemble approaches such as Boosting and Bagging ([Opitz and Shavlik, 1996] compares with Bagging, but not Boosting).

Compared to boosting, which requires a "weak" base learner that does not completely fit the training data (boosting terminates once it constructs a hypothesis with zero training error), DECORATE requires a strong learner, otherwise the artificial diversity training data may prevent it from adequately fitting the real data. When applying boosting to strong base learners, they must first be appropriately weakened in order to benefit from boosting. Therefore, DECORATE may be a preferable ensemble meta-learner for strong learners.

To our knowledge, the only other ensemble approach to utilize artificial training data is the active learning method introduced in [Cohn *et al.*, 1994]. The goal of the committee here is to select good new training examples rather than to improve accuracy using the existing training data. Also, the labels of the artificial examples are selected to produce hypotheses that more faithfully represent the entire version space rather than to produce diversity. Cohn's approach labels artificial data either all positive or all negative to encourage, respectively, the learning of more general or more specific hypotheses.

6 Future Work and Conclusion

In our current approach, we are encouraging diversity using artificial training examples. However, in many domains, a large amount of unlabeled data is already available. We could exploit these unlabeled examples and label them as diversity data. This would allow DECORATE to act as a form of *semi-supervised learning* that exploits both labeled and unlabeled data [Nigam *et al.*, 2000].

Our current study has used J48 as a base learner; however, we would expect similarly good results with other base learners. Decision-tree induction has been the most commonly used base learner in other ensemble studies, but there has been some work using neural networks and naive Bayes [Bauer and Kohavi, 1999; Opitz and Maclin, 1999]. Experiments on "DECORATing" other learners is another area for future work.

By manipulating artificial training examples, DECORATE is able to use a strong base learner to produce an effective, diverse ensemble. Experimental results demonstrate that the approach is particularly effective at producing highly accurate ensembles when training data is limited, outperforming both bagging and boosting low on the learning curve. The empirical success of DECORATE raises the issue of developing a sound theoretical understanding of its effectiveness. In gen-

Table 3: DECORATE vs AdaBoost

Dataset	1%	2%	5%	10%	20%	30%	40%	50%	75%	100%
anneal	**75.29**/73.02	78.14/77.12	85.24/**87.51**	92.26/**94.16**	96.48/**97.13**	97.36/**97.95**	97.73/**98.54**	98.16/**98.8**	98.39/**99.23**	98.71/**99.68**
audio	16.66/16.66	23.73/23.41	**41.72**/40.24	**55.42**/52.7	64.09/64.15	67.62/68.91	70.46/**73.07**	72.82/**75.92**	77.8/**81.74**	82.1/**84.52**
autos	24.33/24.33	29.6/29.71	36.73/34.2	42.89/43.28	52.2/**56.13**	59.86/**62.2**	64.77/**69.14**	68.6/**72.03**	78/**80.28**	83.64/**85.28**
breast-w	**92.38**/74.73	**94.12**/87.84	**95.06**/91.15	**95.64**/93.75	**95.55**/94.85	95.91/95.72	96.2/95.84	96.01/95.87	96.28/96.3	96.31/96.47
credit-a	**71.78**/68.8	74.83/75.3	80.61/79.68	**83.09**/81.14	**84.38**/83.04	84.68/84.22	**85.22**/84.13	**85.57**/84.58	85.61/84.93	85.93/85.42
Glass	31.69/31.69	**35.86**/32.93	**44.5**/40.71	**55.4**/49.78	**61.77**/58.03	66.01/64.33	68.07/66.93	68.85/68.69	72.73/**74.69**	72.77/**76.06**
heart-c	**58.66**/49.57	**65.11**/58.65	**73.55**/70.71	**75.05**/72.5	77.66/76.65	78.34/78.26	79.09/78.96	79.46/79.55	78.74/79.06	78.48/79.22
hepatitis	52.33/52.33	**72.14**/65.93	**76.8**/73.01	**79.48**/76.95	80.7/79.44	**81.81**/79.22	81.65/81.27	83.19/82.63	82.99/83.24	82.62/82.71
colic	**58.37**/52.85	66.58/67.18	**75.85**/72.85	**79.54**/77.17	**81.33**/79.36	**82.47**/79.24	**83.02**/79.51	**83.1**/80.22	**84.02**/80.59	**84.69**/81.93
iris	33.33/33.33	**50.27**/33.33	**80.67**/66.2	**91.53**/84.53	**93.2**/90.73	**94.2**/93	**94.73**/93.33	**94.4**/93.53	94.53/94.2	94.67/94.2
labor	54.27/54.27	54.27/54.27	**67.63**/58.93	**70.23**/65.1	**79.77**/73.2	**83**/76.9	**84.17**/79.57	**83.43**/80.1	**89.73**/84.07	**89.73**/86.37
lymph	48.39/48.39	**53.62**/46.64	**65.06**/60.54	71.2/69.57	**76.74**/74.16	78.84/78.62	78.17/**80.35**	78.99/79.88	79.14/**80.96**	79.08/**81.75**
segment	**67.03**/60.22	**81.16**/77.38	**89.61**/88.5	92.83/92.71	94.88/95.01	95.94/96.03	96.47/**96.9**	96.93/**97.23**	97.58/**98**	98.03/**98.34**
soybean	**19.51**/14.26	**32.4**/23.36	**55.36**/49.37	**73.06**/69.49	85.14/85.01	88.27/88.37	90.22/90.04	91.4/90.89	92.75/92.57	**93.89**/92.88
splice	62.77/**65.11**	67.8/**73.9**	77.37/**82.22**	82.55/**86.13**	88.24/88.27	90.47/89.82	**91.84**/90.8	**92.41**/90.78	**93.44**/92.63	93.92/93.59
Win/Draw/Loss	14/0/1	11/0/4	13/0/2	12/0/3	10/0/5	10/0/5	10/0/5	9/0/6	6/0/9	6/0/9
Sig. W/D/L	7/7/1	8/6/1	11/2/2	10/3/2	7/6/2	4/9/2	5/5/5	5/6/4	3/6/6	3/6/6
GM error ratio	0.8812	0.8937	0.8829	0.9104	0.9407	0.9598	0.9908	0.9957	1.0377	1.0964

eral, the idea of using artificial or unlabeled examples to aid the construction of effective ensembles seems to be a promising approach worthy of further study.

Acknowledgments

This work was supported by DARPA EELD Grant F30602-01-2-0571.

References

[Bauer and Kohavi, 1999] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36, 1999.

[Blake and Merz, 1998] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/˜mlearn/MLRepository.html, 1998.

[Breiman, 1996] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Cohn *et al.*, 1994] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[Cunningham and Carney, 2000] P. Cunningham and J. Carney. Diversity versus quality in classification ensembles based on feature selection. In *11th European Conference on Machine Learning*, pages 109–116, 2000.

[Dietterich, 2000] T. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2000.

[Freund and Schapire, 1996] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, July 1996.

[Hastie *et al.*, 2001] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Verlag, New York, August 2001.

[Krogh and Vedelsby, 1995] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation and active learning. In *Advances in Neural Information Processing Systems 7*, 1995.

[Kuncheva and Whitaker, 2002] L. Kuncheva and C. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *submitted*, 2002.

[Liu and Yao, 1999] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12, 1999.

[Nigam *et al.*, 2000] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39:103–134, 2000.

[Opitz and Maclin, 1999] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[Opitz and Shavlik, 1996] D. Opitz and J. Shavlik. Actively searching for an effective neural-network ensemble. *Connection Science*, 8, 1996.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo,CA, 1993.

[Quinlan, 1996] J. Ross Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence*, August 1996.

[Rosen, 1996] B. Rosen. Ensemble learning using decorrelated neural networks. *Connection Science*, 8, 1996.

[Webb, 2000] G. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40, 2000.

[Witten and Frank, 1999] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, San Francisco, 1999.

[Zenobi and Cunningham, 2001] G. Zenobi and P. Cunningham. Using diversity in preparing ensembles of classifiers based on different feature subsets to minimize generalization error. In *Proceedings of the European Conference on Machine Learning*, 2001.

Proceedings of the KDD-2003 Workshop on Multi-Relational Data Mining
(MRDM-2003), pp.107-121, Washington DC, August, 2003

Scaling Up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism

Lappoon R. Tang, Raymond J. Mooney, and Prem Melville

Department of Computer Sciences,
University of Texas at Austin,
Austin, TX 78712, U.S.A.
{rupert, mooney, melville}@cs.utexas.edu
http://www.cs.utexas.edu/users/ml/

Abstract. Inductive Logic Programming (ILP) has been shown to be a viable approach to many problems in multi-relational data mining (e.g. bioinformatics). Link discovery (LD) is an important task in data mining for counter-terrorism and is the focus of DARPA's program on Evidence Extraction and Link Discovery (EELD). Learning patterns for LD is a novel problem in relational data mining that is characterized by having an unprecedented number of background facts. As a result of the explosion in background facts, the efficiency of existing ILP algorithms becomes a serious limitation. This paper presents a new ILP algorithm that integrates top-down and bottom-up search in order to reduce search when processing large examples. Experimental results on EELD data confirm that it significantly improves efficiency over existing ILP methods.

1 Introduction

The terrible events of September 11, 2001 have sparked increased development of information technology that can aid intelligence agencies in detecting and preventing terrorism. The Evidence Extraction and Link Discovery (EELD) program of the Defense Advanced Research Projects Agency (DARPA) is one attempt to develop new computational methods for addressing this problem. More precisely, *Link Discovery* (LD) is the task of identifying known, complex, multi-relational patterns that indicate potentially threatening activities in large amounts of relational data. Some of the input data for LD comes from *Evidence Extraction* (EE), which is the task of obtaining structured evidence data from unstructured, natural-language documents (e.g. news reports), other input data comes from existing relational databases (e.g. financial and other transaction data). Finally, *Pattern Learning* (PL) concerns the automated discovery of new relational patterns for detecting potentially threatening activities in large amounts of multi-relational data.

Scaling to large datasets in data mining typically refers to increasing the *number* of training examples that can be processed. Another measure of complexity that is particularly relevant in multi-relational data mining is the *size* of

examples, by which we mean the number of ground facts used to describe the examples. To our knowledge, the challenge problems developed for the EELD program are the largest ILP problems attempted to date in terms of the number of ground facts in the background knowledge. Relational data mining in bioinformatics [5] was probably the previously largest ILP problem in this sense. Table 1 shows a comparison between link discovery and, to our knowledge, the largest problem in bioinformatics.

Domain	*# Bg. preds.*	*Avg. Arity*	*# Bg. facts*
Link Discovery	52	2	$\approx$ 568k
Bioinformatics	36	4.9	$\approx$ 24k

Table 1. Link Discovery versus Bioinformatics (e.g. carcinogenesis). *# Bg. preds.* is the number of different predicate names in the background knowledge, *Avg. Arity* is the average arity of the background predicates, and *# Bg facts* is the total number of ground background facts.

Scaling up ILP to efficiently process large examples like those encountered in EELD is a significant problem. Section 2 discusses the problems existing ILP algorithms have scaling to large examples and presents our general approach to controlling the search for multi-relational patterns by integrating top-down and bottom-up search. Section 3 presents the details of our new algorithm, BETH. Section 4 presents some theoretical results on our approach. Experimental results are presented and discussed in Section 5, followed by concluding remarks in Section 6.

2 Combining Top-down and Bottom-up Approaches in BETH

The two standard approaches to ILP are bottom-up and top-down [6]. Bottom-up methods start with a very specific clause generated from an individual positive example and generalize it as far as possible without covering negative examples. Top-down methods start with the most general (empty) clause and repeatedly specialize it until it no longer covers negative examples. Both approaches have problems scaling to large examples.

The state-of-the-art ILP approach, originated from bottom-up methods, is based on inverse entailment [2]. The most popular approach to implementing inverse entailment is a two-stage process: 1) *saturation* which builds up the most specific clause (a.k.a. *bottom clause*) describing a positive example, and 2) *truncation* which finds solutions that generalize the bottom clause [3]. This approach is implemented in PROGOL [2] and its successor ALEPH.[1]

[1] The Aleph Manual can be accessed via `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html`.

Given a positive example and background knowledge, the bottom clause can be infinite, and practically one has to bound it. In PROGOL, it is bounded by five parameters: i, r, $\mathcal{M}$, j^-, and j^+ (please refer to [2] for more details). Unfortunately, the complexity of PROGOL's bottom clause is exponential w.r.t. the variable depth i, which results in a hypothesis space that is doubly exponential! (The size of the subsumption lattice is two to the power of the size of the bottom clause.)

In problems with large examples like EELD, the background knowledge contains many facts using numerous predicates that describe each complex object or event. Typically, many of these facts are irrelevant to the task. However, PROGOL's bottom clause includes every piece of background knowledge (within the recall bound r) in its body. This leads to intractably large bottom-clauses which generates an exponentially larger hypothesis space when learning a clause. This leads one to wonder if it is possible to bound the bottom-clause differently so that it contains only a relevant subset of background facts.

A strength of the top-down approach is that the generation of literals is inherently directed by the heuristic search process itself: only the set of literals that make refinements to clauses in the search beam are generated. Clauses with insufficient heuristic value are discarded, saving the need to generate literals for them. So, there is a tangible link between the entire set of literals that could be included in a bottom-clause and the heuristic search for a good clause. Therefore, perhaps it is possible to employ the heuristic search as a guide to selecting a relevant subset of background facts for inclusion in an alternative bottom-clause.

A major weakness of the top-down approach (as far as literal generation is concerned) is that the enumeration of all possible combination of variables generates many more literals than necessary; some literals generated by the algorithm are not even guaranteed to cover one positive example. The complexity of enumerating all such combinations in FOIL [7] (and mFOIL [6]) is exponential w.r.t. the arity of the predicates [8]. [2] A corresponding strength of the bottom-up approach is that a literal is created using a ground atom describing a known positive example. The advantages are: 1) specializing using this literal results in a clause that is guaranteed to cover at least the seed example, and 2) the set of literals generated are constrained to those that satisfy 1).

Given the strengths and weaknesses of typical top-down and bottom-up approaches, it seems that one can take advantage of the strength of each approach by combining them into one coherent approach. More precisely, we no longer build the bottom clause using a random seed example before we start searching for a good clause. Instead, after a seed example is chosen, one generates literals in a top-down fashion (i.e. guided by heuristic search) except that the literals generated are constrained to those that cover the seed example. Based on this idea, we have developed a new system called **B**ottom-clause **E**xploration

[2] Enforcing argument type restrictions can help lower the complexity but cannot completely solve the problem.

Through **H**euristic-search (BETH) in which the bottom clause is not constructed in advance but "discovered" during the search for a good clause.[3]

3 The Algorithm

BETH's bottom clause is virtual in the sense that the algorithm does not have to construct it to work, unlike PROGOL/ALEPH; it is, nonetheless, constructed to facilitate collection of statistics. However, the virtual bottom clause is a real bound on the subsumption lattice (see Section 4).

Let us begin with some basic definitions. A *predicate specification* takes the form *PredName/Arity* where *PredName* is the name of the predicate in concern and *Arity* its arity. A list of predicate specifications for the background knowledge is given to the ILP system before learning starts. The function $predname(P)$ returns the predicate name of the predicate specification of the background predicates P and $arity(P)$ returns its arity. Likewise, $predname(L)$ returns the predicate name of the literal L and $arity(L)$ returns its arity.

3.1 Constructing a Clause

The outermost loop of BETH is a simple set covering algorithm like that of any typical ILP algorithm: 1) find a good clause which covers a non-empty subset of positive examples, 2) remove the positive examples covered by the clause from the entire set of positive examples, 3) add the clause found to the set of clauses being built (a.k.a. theory) which was initially empty, 4) repeat step 1) to step 3) until the entire set of positive examples are covered by the theory, 5) return the entire set of clauses found.

The way a clause is constructed in BETH is very similar to a traditional top down ILP algorithm like FOIL; the search for a good clause goes from general to specific. It starts with the most general clause $\square$ which is specialized by adding a literal to its body. The most general clause $\square = T \leftarrow true$ where T is a literal such that $predname(T) = predname(e)$ and $arity(T) = arity(e)$, where e is a randomly chosen seed example from the set of positive examples. A beam of potentially good clauses is kept while searching for refinements of each clause in the beam. The construction of a clause terminates when there is a clause in the beam which is sufficiently accurate (i.e. its m-estimate is greater than or equal to a certain threshold).

In addition, we also compute the bottom clause which bounds the search space. The initial bottom clause is set to the smallest (i.e. $e \leftarrow true$ which has an empty set of literals in the body of this initial bottom clause and e is from the set of positive examples) in which case the search space contains only the most general clause (a.k.a. the empty clause). The bound is expanded incrementally during the search for a good clause. The bound is fixed when a *sufficiently good*

[3] PROGOL and ALEPH are really, more precisely, "Subsumption lattice exploration through heuristic-search". Here, we explore the bottom clause and the subsumption lattice simultaneously.

clause is found, at which point both the clause and the bound are returned as solutions to the search. The algorithm which constructs a clause is outlined in Figure 1.

1. Given a set of predicate specifications $\mathcal{P}$ of the background predicates, a beam width b, a bound on the clause length n, variable depth bound i, recall bound r, a non-empty set of positive examples Pxs and a set (possibly empty) of negative examples Nxs.
2. Randomly choose a seed example $e \in Pxs$.
3. $\bot_0 := e \leftarrow true$.
4. $Q_0 := \{\square\}$.
5. $Q := Q_0$.
6. $\bot := \bot_0$.
7. REPEAT
   $generate\_refinements(Q, \mathcal{P}, b, n, i, r, Pxs, Nxs, Q', \bot, \bot')$,
   $Q := Q'$,
   $\bot := \bot'$
   UNTIL
   there is a clause $C \in Q$ which is *sufficiently* accurate. Q' and $\bot'$ are output variables and the rest in $generate\_refinements$ are input variables.
8. Return C and $\bot$.

Fig. 1. The construction of a clause in BETH

3.2 Generating Refinements for a Clause

To find all the refinements of a given clause C_i, first find a substitution θ, which satisfies the body of the clause (a "successful proof" of the clause, given the background facts); then construct a literal (with dummy variables) R_j for a predicate specification in $\mathcal{P}$, and find a substitution β that makes $R_j\beta$ a ground atom such that $C_i\theta$ and $R_j\beta$ satisfy the following conditions we call *refinement constraints*: 1) the link constraint: one of the arguments of $R_j\beta$ has to appear in $C_i\theta$ (this is to make sure that the resulting clause is still a linked clause), 2) the unique-literal constraint: $R_j\beta \notin C_i\theta$ (to avoid making two identical literals). We try to find pairs of θ and β satisfying the refinement constraints, but at most r distinct ground atoms $R_j\beta$ will be used. For example, suppose $C_i\theta = f(a,b) \leftarrow g(a,c), h(c,d), g(b,e)$ and $R_j\beta_1 = g(e,f)$ and $R_j\beta_2 = g(a,c)$, then only $R_j\beta_1$ satisfies all the refinement constraints, as $R_j\beta_2$ fails the unique-literal constraint. So, only $R_j\beta_1$ will be used to make literals for the clause C_i.

To avoid repeatedly finding a successful proof of a given clause by theorem proving, we make a set of "cached proofs" for each clause in the beam (similar to the way variable bindings are stored in extensional FOIL) by starting with the initial proof $e \leftarrow true$, where e is a randomly chosen seed example, and we incrementally update the cache of proofs of each clause by adding to the end of each proof a ground atom satisfying all the refinement constraints. A bound is also given to the cache size. When finding a satisfying substitution θ for a clause C_i in the beam, we will simply unify C_i with a proof in its cache. If there is no $R_j\beta$ satisfying the refinement constraints, which can happen if the first chosen example e was a "bad" one, a new example $e' \neq e$ will be randomly chosen

from the remaining set of positive examples to be covered. The clause C_i will be replaced (in the beam) by the most general clause such that its cache of proofs will contain only $e' \leftarrow true$. The idea is that if a clause cannot be refined, then we will just restart with a different seed example.

One can also take advantage of type declarations (if available) to further restrict the number of predicate specifications needed to be considered for a given clause — one needs only to consider those which contain at least one argument type which is the same as at least one of the types of all the variables in the current clause (so that a linked clause that satisfies the type constraints is possible).

One can also make use of mode declarations (if available) by substituting arguments with "input" mode for constants which appear in the clause, provided that the argument type and the constant type are the same (similar to the way the bottom clause is built in PROGOL). One needs to find satisfying substitutions for R_j, for each unique way of substituting arguments with input mode for constants in the clause. The algorithm for generating refinements to a clause is outlined in Figure 2.

1. Given a set of predicate specifications $\mathcal{P}$ of the background predicates, a beam width b, a bound on the clause length n, variable depth bound i, recall bound r, a non-empty set of positive examples Pxs and a set (possibly empty) of negative examples Nxs, the current bottom clause $\perp$ (i.e. the current bound on the search space).
2. For each clause $C_i \in Q$ and for each $P_j \in \mathcal{P}$, make a literal R_j with dummy variables such that $predname(R_j) = predname(P_j)$ and $arity(R_j) = arity(P_j)$.
3. Find substitutions θ, β such that 1) θ satisfies C_i, 2) β satisfies R_j, and 3) $C_i\theta$ and $R_j\beta$ satisfy all the *refinement constraints*.
4. Collect at most r such ground atoms $R_j\beta$ for different θ and β.
5. For each pair of $C_i\theta$ and $R_j\beta$ satisfying all the refinement constraints, $make\_literals(C_i\theta, R_j\beta, Lits)$ and add $R_j\beta$ to the body of $\perp$.
6. For each $L \in Lits$, add L to the body of C_i to make C_i' and let the set of all C_i''s be Q_i.
7. Evaluate each clause in $\bigcup_{C_i \in Q} C_i$ by a heuristic (e.g. m-estimate) given Pxs and Nxs.
8. Put only the best b clauses into Q'.
9. Let $\perp'$ be the resulting bottom clause after adding all the ground atoms $R_j\beta$'s to the body of $\perp$ for each C_i and R_j (such that there exists θ and β satisfying all the refinement constraints).
10. Return Q' and $\perp'$.

Fig. 2. Generate Refinements

3.3 Making Literals

To make literals given a clause C, a satisfying substitution θ of C, and a ground atom $R_j\beta$, we replace arguments of $R_j\beta$ by variables in C instantiated, in θ, to these arguments in $R_j\beta$, *only if* $R_j\beta$ is not in $C\theta$ and the resulted literal observes the variable depth bound. θ^{-1} replaces all occurrences of a term by the same variable. For example, consider a clause $C : f(A, B) \leftarrow g(B, D), h(A, E), l(D, E)$, $\theta = \{A/a, B/b, D/b, E/e\}$ (thus, $\theta^{-1} = \{a/A, b/B, b/D, e/E\}$) and two ground atoms $a_1 = p(b, e)$ and $a_2 = p(e, f)$. We can make two literals using a_1: 1) $p(B, E)$ (since $b/B, e/E \in \theta^{-1}$), and 2) $p(D, E)$ (since $b/D, e/E \in \theta^{-1}$). We can

make one literal using a_2: $p(E, F)$ (since $e/E \in \theta^{-1}$, but the constant f is not bound to any variable in θ^{-1}). However, if the variable depth bound is one, then the literal $p(E, F)$ will be rejected because the depth of F is two. The variable depth $d(V)$ of variable V is defined in LINUS [6]. The algorithm for making literals is outlined in Figure 3.

1. Given a clause $C\theta$ of the form $e \leftarrow a_1, \ldots, a_n$ (where C is the current clause being refined, i.e. specialized, and θ is a substitution that satisfies C and $e \in Pxs$ and background knowledge $BK \models a_i$ for each ground atom a_i in the body of $C\theta$) and a ground atom a_{n+1} such that $BK \models a_{n+1}$.
2. Make a set of literals $Lits$ such that each literal $L \in Lits$ satisfies: 1) $predname(L) = predname(a_{n+1})$, 2) $arity(L) = arity(a_{n+1})$, 3) suppose the constant c_i is the ith argument of a_{n+1} and the variable V_i is the ith argument of L. If c_i appears in $C\theta$, then $c_i/V_i \in \theta^{-1}$; otherwise, V_i is a new variable not appearing in C, 4) there is no variable V in L such that $d(V) > i$ where i is the variable depth bound.
3. Return $Lits$.

Fig. 3. Make Literals

3.4 A Concrete Example

We can see how the algorithm works through a simple example from the family-relation domain. Suppose we want to learn the concept $uncle(X, Y)$, which is true iff X is an uncle of Y (blood uncle).

Suppose we have the following set of background facts (Figure 4):

1. $male(Bob)$, $male(Tom)$, $male(Tim)$
2. $female(Ann)$, $female(Mary)$, $female(Susan)$, $female(Betty)$, $female(Joyce)$
3. $parent(Tom, Mary)$, $parent(Tom, Betty)$, $parent(Tom, Bob)$, $parent(Mary, Ann)$,$parent(Joyce, Susan)$,$parent(Tom, Tim)$
4. $friend(Mary, Susan)$, $friend(Susan, Mary)$, $friend(Joyce, Betty)$, $friend(Betty, Joyce)$

and $\mathcal{P} = \{male/1, female/1, friend/2, parent/2\}$ (exactly in this order from left to right) is our set of predicate specifications. We will use $'+'$ to denote the output mode and $'-'$ the input mode here. The following is the set of mode specifications for each predicate specification:

$$male(-), female(-), parent(+,-), parent(-,+), friend(+,-), friend(-,+)$$

and the following is the set of type specifications for each predicate specification:

$$male(person), female(person), parent(person, person), friend(person, person)$$

Suppose we have this set of training examples:

1. Positive: $uncle(Bob, Ann)$
2. Negative:
   $uncle(Bob, Susan)$, $uncle(Betty, Ann)$, $uncle(Tim, Susan)$, $uncle(Tom, Betty)$
   $uncle(Susan, Betty)$, $uncle(Joyce, Ann)$, $uncle(Tim, Joyce)$, $uncle(Tom, Mary)$

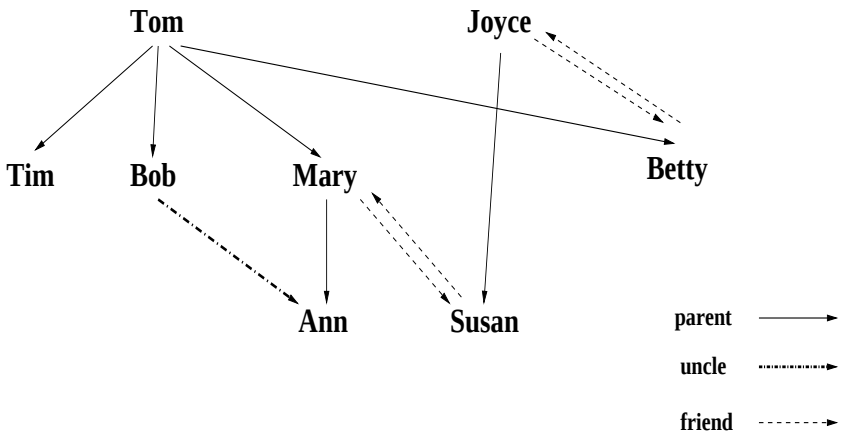


Fig. 4. A simple family relation domain

We present a trace of how our algorithm discovers a good clause, given a beam size and a recall bound of one, and a clause length of four. It starts by choosing a random seed example from the set of positive examples. This has to be $uncle(Bob, Ann)$ since there is only one positive example. When generating refinements to a clause, it considers each predicate specification in $\mathcal{P}$ (from left to right). We will show the specialized clause before its set of cached proofs. The literal added to the clause currently being built is generated from the *new* ground atom added to the body of the cached proof of the current clause.

The algorithm starts with:

1. The most general clause which covers every pair of people: `uncle(X,Y) :- true`
2. The set of cached proofs for this clause: {`uncle(bob,ann) :- true`}
3. The empty bottom clause: `uncle(bob,ann) :- true`

It considers $male/1$ and generates the following:

1. The specialized clause: `uncle(X,Y) :- male(X)`
   `(m-est = 0.153)`
2. The set of cached proofs for this clause: {`uncle(bob,ann) :- male(bob)`}
3. The updated bottom clause: `uncle(bob,ann) :- male(bob)`

Next, the algorithm considers $female/1$, and the literal `female(Y)` is generated (in the same way as $male/1$), the new ground atom `female(ann)` is added to the current bottom clause. The specialized clause `uncle(X,Y) :- female(Y)` has an m-estimate of 0.111. Next, it considers $parent/2$ (using $parent(+,-)$) and generates the following:

1. The specialized clause: `uncle(X,Y) :- parent(Z,X)`
   `(m-est = 0.136)`

2. The set of cached proof of this clause: {`uncle(bob,ann) :- parent(tom,bob)`}
3. The updated bottom clause:
   `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob)`

Similarily $parent/2$ (using $parent(+,-)$) is used to generate another specialized clause `uncle(X,Y) :- parent(W,Y)` (m-estimate = 0.122) using the ground atom `parent(mary,ann)`.

The predicate specification $friend/2$ was considered but no ground atom was found to satisfy all the refinement constraints; the link constraint could not be satisfied, because neither Bob nor Ann has a friend. There are totally four different refinements to the most general clause. The clause with the best m-estimate is:

$$uncle(X,Y) \leftarrow male(X)$$

Since the beam size is just one, only this clause is retained in the beam. This clause is still covering negative examples: $uncle(Bob, Susan)$, $uncle(Tom, Betty)$, $uncle(Tim, Susan)$, $uncle(Tim, Joyce)$, and $uncle(Tom, Mary)$. So, it still needs to be refined. Next, $male/1$ is considered but no ground atom is found to satisfy all the refinement constraints; the unique-literal constraint could not be satisfied ($male(Bob)$ is already in the cached proof of the clause). The current bottom clause is `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob)`.

Next, it considers $female/1$ and generates the following:

1. The specialized clause: `uncle(X,Y) :- male(X),female(Y)`
   `(m-est = 0.153)`
2. The set of cached proof of this clause:
   {`uncle(bob,ann) :- male(bob),female(ann)`}
3. The updated bottom clause:
   `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),`
   `parent(mary,ann)`

Next, it considers $parent/2$ (using $parent(+,-)$) and generates the following:

1. The specialized clause: `uncle(X,Y) :- male(X),parent(Z,X)`
   `(m-est = 0.204)`
2. The set of cached proof of this clause:
   {`uncle(bob,ann) :- male(bob),parent(tom,bob)`}
3. The updated bottom clause:
   `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),`
   `parent(mary,ann)`

$parent/2$ (using $parent(+,-)$) can be used to generate another specialized clause `uncle(X,Y) :- male(X),parent(W,Y)` (m-estimate = 0.175) using the ground atom `parent(mary,ann)`.

The predicate specification $friend/2$ was considered but no ground atom was found to satisfy all the refinement constraints (the link constraint cannot be satisfied). There are totally three different refinements to $uncle(X,Y) \leftarrow male(X)$. The clause with the best m-estimate is:

$$uncle(X,Y) \leftarrow male(X), parent(Z,X)$$

This clause still covers a non-empty set of negative examples:

$$uncle(Bob, Susan), uncle(Tim, Susan), uncle(Tim, Joyce).$$

The algorithm continues in exactly the same manner for the last two steps (omitted to save space). The clause $uncle(X, Y) \leftarrow male(X), parent(Z, X)$ has four different refinements. The clause with the best m-estimate is:

$$uncle(X, Y) \leftarrow male(X), parent(Z, X), parent(W, Y)$$

which is still covering a non-empty set of negative examples: $uncle(Bob, Susan)$ and $uncle(Tim, Susan)$.

There are totally eight different refinements to

$$uncle(X, Y) \leftarrow male(X), parent(Z, X), parent(W, Y).$$

The clause with the best m-estimate is:

$$uncle(X, Y) \leftarrow male(X), parent(Z, X), parent(W, Y), parent(Z, W)$$

which covers all the positive examples and no negative examples. At this point, the algorithm has found the target concept. Both the bottom clause discovered and the consistent clause found are returned. Notice that the bottom clause found by BETH is:

```
uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),parent(mary,ann),
male(tom),female(mary),parent(tom,mary),friend(mary,susan),
friend(susan,mary)
```

whereas, PROGOL's bottom clause is:

```
uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),parent(mary,ann),
male(tom),female(mary),parent(tom,mary),friend(mary,susan),
friend(susan,mary),female(susan)
```

which is bigger than BETH's bottom clause.

4 Analysis

Let $\perp(b, n, \mathcal{P}, r, i)$ be the bottom clause constructed by BETH (Section 3) given the parameters b, n, $\mathcal{P}$, r, and i which are the beam width, the maximum clause length, the set of predicate specifications, the recall bound, and the variable depth bound respectively.

Theorem 1. Suppose B is a beam of clauses produced by BETH, for any clause $C \in B$, $C \preceq \perp(b, n, \mathcal{P}, r, i)$.

Proof. Suppose C_j is a clause in B such that $C_j = H \leftarrow L_1, \ldots, L_m$ where $m \leq n$. Each L_k is produced from a ground atom a_k and H from a particular seed example e. Obviously, $e \in \perp(b, n, \mathcal{P}, r, i)$. For any $k : 1 \leq k \leq m$, $a_k \in \perp(b, n, \mathcal{P}, r, i)$, since each ground atom satisfying all the refinement constraints is added to the current bottom clause and only ground atoms satisfying all the refinement constraints are used to make literals for any clause.

Thus, there's a substitution θ which satisfies C_j s.t. $C_j\theta = e \leftarrow a_1, \ldots, a_m$. So, $C_j\theta \subseteq \bot(b, n, \mathcal{P}, r, i)$. And, we have $C_j \preceq \bot(b, n, \mathcal{P}, r, i)$. Hence we have $C \preceq \bot(b, n, \mathcal{P}, r, i)$ for any clause $C \in B$. □

Theorem 2. The worst case length of $\bot(b, n, \mathcal{P}, r, i)$ is $\mathcal{O}(bn|\mathcal{P}|r)$.

Proof. The maximum number of ground atoms that 1) satisfy the refinement constraints and 2) make literals observing the variable depth bound i for any clause in the search beam at the point the clause is being refined are $|\mathcal{P}|r$. Therefore, the maximum number of ground atoms satisfying the refinement constraints after adding n literals to the body of the most general clause are $n|\mathcal{P}|r$. Since there are at most b clauses in the search beam at any time, the maximum number of ground atoms satisfying the refinement constraints are $bn|\mathcal{P}|r$. Thus, the worst case complexity of the bottom clause $\bot(b, n, \mathcal{P}, r, i)$ is $\mathcal{O}(bn|\mathcal{P}|r)$. □

The length of PROGOL's bottom clause is $\mathcal{O}((r|\mathcal{M}|j^+j^-)^{ij^+})$; where $|\mathcal{M}|$ is the number of mode declarations, and $j^{+/-}$ are bounds on the number of $(+/-)$ types in a mode declaration [2] — which makes a hypothesis space doubly exponential w.r.t. i. Whereas the length of BETH's bottom clause is only linear w.r.t. n (which gives rise to a much smaller hypothesis space).

5 Experimental Evaluation

We compared our system, BETH, with two other leading ILP systems — ALEPH and mFOIL.

5.1 Domain

After the events of 9/11, the EELD project has been working on several Challenge Problems that are related to counter-terrorism. The problem that we choose to tackle is the detection of Murder-For-Hires (contract killings) in the domain of Russian Organized Crime. The data used in all EELD Challenge Problems include representations of people, organizations, objects, and actions and many types of relations between them. One can picture this data as a large graph of entities connected by a variety of relations. For our purposes, we represent these relational databases as facts in Prolog.

For the ease of generating large quantities of data, and to avoid violating privacy, the program currently only uses synthetic data generated by a simulator. The data for the Murder-For-Hire problem was generated using a Task-Based (TB) simulator developed by Information Extraction and Transport Incorporated (IET). The TB simulator outputs case files, which contain complete and unadulterated descriptions of murder cases. These case files are then filtered for observability, so that facts that would not be accessible to an investigator are eliminated. To make the task more realistic, this data is also corrupted, e.g., by misidentifying role players or incorrectly reporting group memberships. This filtered and corrupted data form the evidence files. In the evidence files, facts about each event are represented as ground facts, such as:

```
murder(Murder714)
```

```
perpetrator(Murder714, Killer186)
crimeVictim(Murder714, MurderVictim996)
deviceTypeUsed(Murder714, PistolCzech)
```

The synthetic dataset that we used consists of 632 murder events. Each murder event has been labeled as either a positive or negative example of a murder-for-hire. There are 133 positive and 499 negative examples in the dataset. Our task was to learn a theory to correctly classify an unlabeled event as either a positive or negative instance of murder-for-hire. The amount of background knowledge for this dataset is extremely large; consisting of 52 distinct predicate names, and 681,039 background facts in all.

5.2 Results

The performance of each of the ILP systems was evaluated using 6-fold cross-validation. The total number of Prolog atoms in the data is so large that running more than six folds is not feasible.[4] The data for each fold was generated by separate runs of the TB simulator. The facts produced by one run of the simulator, only pertain to the entities and relations generated in that run; hence the facts of each fold are unrelated to the others. For each trial, one fold is set aside for testing, while the remaining data is *combined* for training. To test performance on varying amounts of training data, learning curves were generated by testing the system after training on increasing subsets of the overall training data. Note that, for different points on the learning curve, the background knowledge remains the same; only the number of positive and negative training examples given to the system varies.

We compared the three systems with respect to accuracy and training time. Accuracy is defined as the number of correctly classified test cases divided by the total number of test cases. The training time is measured as the CPU time consumed during the training phase. All the experiments were performed on a 1.1 GHz Pentinum with dual processors and 2 GB of RAM. BETH and mFOIL were implemented in Sicstus Prolog version 3.8.5 and ALEPH was implemented in Yap version 4.3.22. Although different Prolog compilers were used, the Yap Prolog compiler has been demonstrated to outperform the Sicstus Prolog compiler, particularly in ILP applications [4].

In our experiments, we used a beam width of 4 for BETH and mFOIL; and limited the number of search nodes in ALEPH to 5000. We used m-estimate ($m = 2$) as a search heuristic for all ILP algorithms. The clause length was limited to 10 and the variable depth bound to 5 for all systems. The recall bound was limited to 1 for BETH and ALEPH (except for some mode declarations it was set to '*'). We modified mFOIL to be constrained by the maximum clause length and the variable depth bound, to ensure that it terminates. We refer to this version of mFOIL as *Bounded* mFOIL. All the systems were given 1 second of CPU time to compute the set of examples covered by a clause. If a specialized

[4] The maximum number of atoms that the Sicstus Prolog compiler can handle is approximately a quarter million.

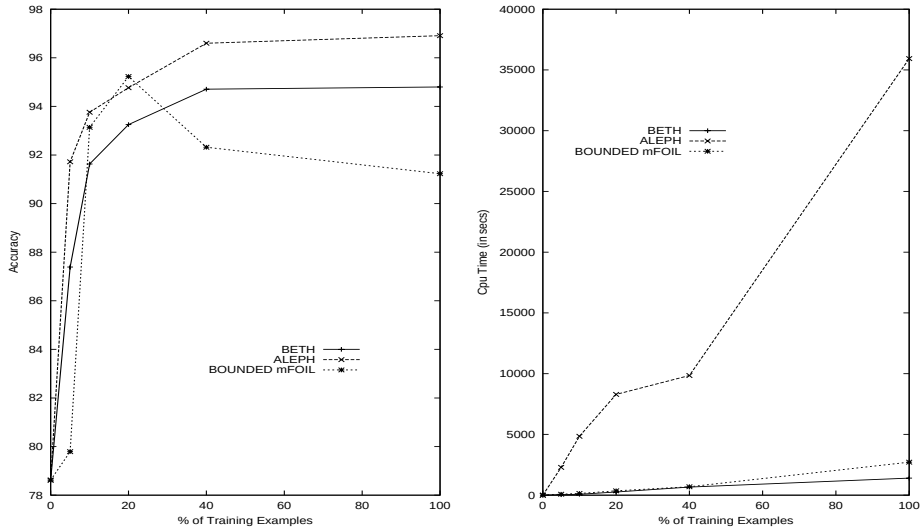


Fig. 5. Performance of the systems versus the percentage of training examples given

System	*Accuracy*	*CPU Time (mins)*	*# of Clauses*	*Bottom Clause Size*
Beth	94.80% (+/- 2.3%)	23.39 (+/- 4.26)	4483	34
Aleph	96.91% (+/- 2.8%)	598.92 (+/- 250.00)	63334	4061
mFoil	91.23% (+/- 4.8%)	45.28 (+/- 5.40)	112904	n/a

Table 2. Results on classifying *murder-for-hire* events given all the training data. *# of Clauses* is the total number of clauses tested; and *Bottom Clause Size* is the average number of literals in the bottom clause constructed for each clause in the learned theory. The 90% confidence intervals are given for test *Accuracy* and *CPU time*.

clause took more time than allotted, the clause was ignored; although the time it took to create the clause is still recorded.

The results of our experiments are summarized in Figure 5. A snapshot of the performance of the three ILP systems given 100% of the training examples is shown in Table 2. The following is a sample rule learned by Beth:

```
murder_for_hire(A):- murder(A), eventOccursAt(A,H),
  geographicalSubRegions(I,H), perpetrator(A,B),
  recipientOfinfo(C,B), senderOfinfo(C,D), socialParticipants(F,D),
  socialParticipants(F,G), payer(E,G), toPossessor(E,D).
```

This rule covered 9 positive examples and 3 negative examples. The rule can be interpreted as: A is a murder-for-hire, if A is a murder event, which occurs in a city in a subregion of Russia, and in which B is the perpetrator, who received information from D, who had a meeting with and received some money from G.

5.3 Discussion of Results

On the full training set, BETH trains 25 times faster than ALEPH while losing only 2 percentage points in accuracy and it trains twice as fast as mFOIL while gaining 3 percentage points in accuracy. Therefore, we believe that its integration of top-down and bottom-up search is an effective approach to dealing with the problem of scaling ILP to large examples. The learning curves further illustrate that the training time of BETH grows slightly slower than that of mFOIL, and considerably slower than that of ALEPH.

The large speedup over ALEPH is explained by the theoretical analysis on the complexity of the bounds on the search space, i.e. the different sizes of the bottom clauses they construct. The size of the bottom clause for BETH is only linear w.r.t. n compared to that of ALEPH which is exponential w.r.t. to i ($i \leq n$) even for small i. As a result, ALEPH's search space is much larger than BETH's. ALEPH's bottom clause was on average 119x larger than BETH's and the total number of clauses it constructed was 14x larger, although a theory of similar accuracy was learned.

Systems like BETH and ALEPH construct literals based on actual ground atoms in the background knowledge, guaranteeing that the specialized clause covers at least the seed example. On the other hand, mFOIL generates more literals than necessary by enumerating all possible combination of variables. Some such combinations make useless literals; adding any of them to the body of the current clause makes specialized clauses that do not cover any positive examples. Thus, mFOIL wastes CPU time constructing and testing these literals. Since the average predicate arity in the EELD data was small (2), the speedup over mFOIL was not as great, although much larger gains would be expected for data that contains predicates with higher arity.

Nevertheless, searching a smaller space comes at the cost of spending more time generating each literal for refining a clause. In ALEPH, all the necessary ground literals are generated before the search starts, while BETH must spend time computing a set of ground atoms satisfying the refinement constraints on literal generation, resulting in fewer clauses tested per unit time compared to both ALEPH and mFOIL.

From the experimental results obtained, we can conclude that 1) an approach like BETH, which emphasizes searching a much smaller space over testing hypotheses at a higher rate, can outperform (in terms of efficiency) an approach like PROGOL/ALEPH, which trades off the two factors the other way around, and 2) using ground atoms directly avoids testing useless literals, improving training time over a purely top-down approach like mFOIL.

6 Conclusions

An important under-studied aspect of scaling to large databases in multi-relational data mining concerns the size of examples rather than their number. For ILP methods, this issue involves scaling to large numbers of connected background

facts associated with each example or set of examples. We have developed a new ILP algorithm that integrates top-down and bottom-up search in order to more efficiently learn in the presence of large sets of background facts. Challenge problems constructed for DARPA's program on Evidence Extraction and Link Discovery concern identifying potential threatening activities in large amounts of heterogeneous, multi-relational data. These problems contain relatively modest numbers of examples but involve very large sets of background facts. Experimental results on these problems demonstrate that our new hybrid approach substantially decreases training time compared to existing ILP methods.

7 Acknowledgments

This research is sponsored by DARPA and managed by Rome Laboratory under contract F30602-01-2-0571. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency, Rome Laboratory, or the United States Government.

References

1. R. J. Mooney, P. Melville, L. R. Tang, J. Shavlik, I. de Castro Dutra, D. Page, and V. S. Costa. Relational data mining with inductive logic programming for link discovery. In *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining*, 2002.
2. S. Muggleton. Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.
3. C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming*, pages 63–92. Academic Press, London, 1992.
4. V. Santos Costa. Optimising bytecode emulation for Prolog. In *LNCS 1702, Proceedings of PPDP'99*, pages 261–267. Springer-Verlag, September 1999.
5. F. Zelezny, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, 2002.
6. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, 1994.
7. R. J. Quinlan. *Learning Logical Definitions from Relations. Machine Learning*, 5(3):239–266, 1990.
8. M. J. Pazzani and D. F. Kibler. *The Utility of Background Knowledge in Inductive Learning. Machine Learning*, 9:57–94, 1992.

CLP($\mathcal{BN}$): Constraint Logic Programming for Probabilistic Knowledge

Vítor Santos Costa
COPPE/Sistemas
UFRJ, Brasil

David Page and Maleeha Qazi
Department of Biostatistics
and Medical Informatics
University of Wisconsin-Madison, USA

James Cussens
Department of Computer Science
University of York, UK

Abstract

In Datalog, missing values are represented by Skolem constants. More generally, in logic programming missing values, or existentially-quantified variables, are represented by terms built from Skolem functors. In an analogy to probabilistic relational models (PRMs), we wish to represent the joint probability distribution over missing values in a database or logic program using a Bayesian network. This paper presents an extension of logic programs that makes it possible to specify a joint probability distribution over terms built from Skolem functors in the program. Our extension is based on constraint logic programming (CLP), so we call the extended language CLP($\mathcal{BN}$). We show that CLP($\mathcal{BN}$) subsumes PRMs; this greater expressivity carries both advantages and disadvantages for CLP($\mathcal{BN}$). We also show that algorithms from inductive logic programming (ILP) can be used with only minor modification to learn CLP($\mathcal{BN}$) programs. An implementation of CLP($\mathcal{BN}$) is publicly available as part of YAP Prolog at http://www.cos.ufrj.br/~vitor/Yap/clpbn.

1 Introduction

A probabilistic relational model (PRM) [4] uses a Bayesian network to represent the joint probability distribution over fields in a relational database. The Bayes net can be used to make inferences about missing values in the database. In Datalog, missing values are represented by Skolem constants; more generally, in logic programming missing values, or existentially-quantified variables, are represented by terms built from Skolem functors. In analogy to PRMs, can a Bayesian network be used to represent the joint probability distribution over terms constructed from the Skolem functors in a logic program? We extend the language of logic programs to make this possible. Our extension is based on constraint logic programming (CLP), so we call the extended language CLP($\mathcal{BN}$). We show that any PRM can be represented as a CLP($\mathcal{BN}$) program.

Our work in CLP($\mathcal{BN}$) has been motivated by our interest in multi-relational data mining, and more specifically in inductive logic programming (ILP). Because CLP($\mathcal{BN}$) programs are a kind of logic program, we can use existing ILP systems to learn them, with only simple modifications to the ILP systems. Induction of clauses can be seen as model generation, and parameter fitting can be seen as generating the CPTs for the constraint of a clause. We show that the ILP system ALEPH [11] is able to learn CLP($\mathcal{BN}$) programs.

This paper is organised as follows. First, we describe the design of CLP($\mathcal{BN}$) through examples. Next, we discuss the foundations of CLP($\mathcal{BN}$), including detailed syntax, proof theory (or operational semantics) and model-theoretic semantics. We then relate CLP($\mathcal{BN}$) with PRMs. Finally, we present the results of experiments in learning CLP($\mathcal{BN}$) programs using ILP.

2 CLP($\mathcal{BN}$) by Example

We shall use the school database scheme originally used to explain Probabilistic Relational Models [4] (PRMs) to guide us through CLP($\mathcal{BN}$). We chose this example because it stems from a familiar background and because it illustrates how CLP($\mathcal{BN}$) relates to PRMs. Figure 1 presents the database scheme and the connection between random variables. There are four relations, describing professors, students, courses, and registrations. Field names in italics correspond to random variables.

Random variables may depend on other random variables, inducing the dependencies shown in Figure 1. In a nutshell, a professor's ability, a course's difficulty, and a student's intelligence do not depend on any other factors explicitly represented in the database. A professor's popularity depends only on his/her ability. A registration's grade depends on the course's difficulty, and on the student's intelligence. A

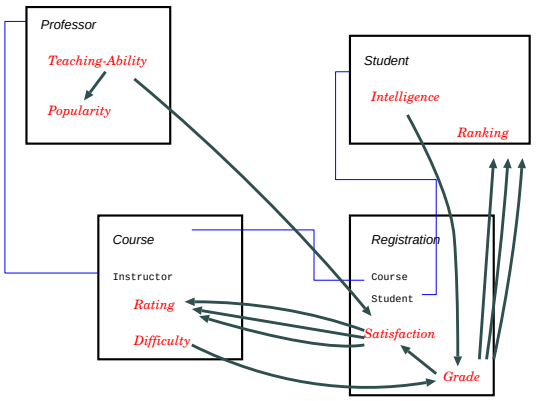


Figure 1: The School Database

student's ranking depends on *all* the grades he/she had, and a course's rating depends on the satisfaction of every student who attended the course.

One possible representation would be to use Skolem functions to represent random attributes. In the simplest case, professor ability, we would write: $\texttt{ability}(\texttt{jim}, skA(\texttt{jim}))$, where $skA(\texttt{jim})$ is a Skolem function of `jim`. Unfortunately, our assertion is not very illuminating. We would also like to represent the probabilities for the different cases of ability, that is, we would like to write:

$$\texttt{ability}(\texttt{jim}, skA(\texttt{jim})) \wedge$$
$$P(skA(\texttt{jim}) = \texttt{h}) = 0.7 \wedge P(skA(\texttt{jim}) = \texttt{l}) = 0.3$$

We would further like to use special inference rules for probabilities. Logic programming systems have used constraints to address similar problems. Logical variables are said to be *constrained* if they are bound to one or more constraints. Constraints are kept in a separate *store* and can be updated as execution proceeds (ie, if we receive new evidence on a variable). Unifying a term with a constrained variable invokes a specialised *solver*. The solver is also activated before presenting the answer to a query. In constraint notation, we could say:

$$\exists X, \quad \texttt{ability}(\texttt{jim}, X) \wedge$$
$$\{X = skA(\texttt{jim}) \wedge P(X = \texttt{h}) = 0.7 \wedge P(X = \texttt{l}) = 0.3\}$$

The curly brackets surround the constraint store: they say that X must take the value of the function $skA(\texttt{jim})$, and that it has two possible values with complementary probabilities. CLP($\mathcal{BN}$) programs manipulate such constraints. We use the following syntax:

```
{Abi = a(jim) with p([h,l],[0.7,0.3],[])}
```

CLP($\mathcal{BN}$) Programs A CLP($\mathcal{BN}$) is a constraint logic program that can encode Bayesian constraints. CLP($\mathcal{BN}$) programs thus consist of clauses. Our first example of a clause defines a student's intelligence in the school database:

```
intelligence(S,Int) :-
 {Int = i(S) with p([h,l],[0.7,0.3],[])}.
```

In this example we have the same information on every student's intelligence. Often, we may have different probability distributions for different students:

```
intelligence(S,Int) :-
  int_table(S, Dist),
  {Int = i(S) with p([h,l],Dist,[])}.

int_table(bob, [0.3, 0.9]) :- !.
int_table(mike, [0.8, 0.2]) :- !.
int_table( _, [0.7,0.3]).
```

Probability distributions are first class objects in our language: they can be specified at compile-time or computed from arbitrary logic programs.

Conditional Probabilities Let us next consider an example of a conditional probability distribution (CPT). In Figure 1 one can observe that a registration's grade depends on the course's difficulty and on the student's intelligence. This is encoded by the following clause:

```
grade(Reg, Grade) :-
  reg(Reg,Course,Student),
  difficulty(Course,Dif),
  intelligence(Student,Int),
  {Grade = grade(Reg) with p(
   [a,b,c],[0.4,0.9,0.4,0.0
            0.4,0.1,0.4,0.1,
            0.2,0.0,0.2,0.9],[Dif,Int])}.
```

To keep the actual CPT small, we assume that Dif and Int each can take only two values. Note that in general, CPTs can be obtained from arbitrary logic programs which compute a number from any *(structured_term,constant)* pair.

Execution The evaluation of a CLP($\mathcal{BN}$) program results in a network of constraints. In the previous example, the evaluation of

```
?- grade(r2,Grade).
```

will set up a constraint network with $\texttt{grade}(\texttt{r2})$ depending on $\texttt{dif}(\texttt{course})$ and $\texttt{int}(\texttt{student})$. CLP($\mathcal{BN}$) will output the marginal probability distribution on $\texttt{grade}(\texttt{r2})$.

One major application of Bayesian systems is conditioning on evidence. For example, if a student had a good grade we may want to find out whether he or she is likely to be intelligent:

```
?- grade(r2,a), intelligence(bob,I).
```

The user introduces evidence for `r2`'s grade through binding the argument in the corresponding goal. In practice, the system preprocesses the clause and adds evidence as an extra constraint on the argument. The system then executes as a standard constraint logic program, building a Bayes

net with two variables. The network is evaluated through variable elimination, or another standard technique.

CLP($\mathcal{BN}$) provides syntactic sugar to insert evidence in the program:

```
grade(r2, a) :- {}.
```

declares that we have evidence that the value of `grade` for registration `r2` is `a`. Compile time evidence is processed at run-time as a query extension.

3 Foundations

We next present the basic ideas of CLP($\mathcal{BN}$) more formally. For brevity, this section necessarily assumes prior knowledge of first-order logic, model theory, and resolution.

First, we remark that CLP($\mathcal{BN}$) programs are logic programs, and thus inherit the well-known properties of logic programs. We further interpret a CLP($\mathcal{BN}$) program as defining a set of probability distributions over the models of the underlying logic program. Any Skolem function sk of variables $X_1, ..., X_n$, has an associated CPT specifying a probability distribution over the possible denotations of $sk(X_1, ..., X_n)$ given the values, or bindings, of X_1, ..., X_n. The CPTs associated with a clause may be thought of as a Bayes net, where each node is labeled by either a variable or a term build from a Skolem function. Figure 2 illustrates this view using a clause that relates a registration's grade to the course's difficulty and to the student's intelligence. At times we will denote a CLP($\mathcal{BN}$) clause by C/B, where C is the logical portion and B is the probabilistic portion. For a CLP($\mathcal{BN}$) program P, the logical portion of P is simply the conjunction of the logical portions of all the clauses in P.

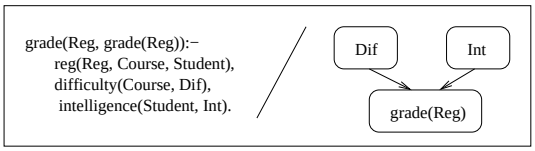


Figure 2: Pictorial representation of a grade clause.

3.1 Detailed Syntax

The alphabet of CLP($\mathcal{BN}$) is the alphabet of logic programs. We shall take a set of functors and call these functors *Skolem functors*; *Skolem constants* are simply Skolem functors of arity 0. A Skolem term is a term whose primary functor is a Skolem functor. We assume that Skolem terms have been introduced into the program during a Skolemization process to replace the existentially-quantified variables in the program. It follows from the Skolemization process that any Skolem functor sk appears in only one Skolem term, which appears in only one clause, though that Skolem term may have multiple occurrences in that one clause. Where the Skolem functor sk has arity n, its Skolem term has the form $sk(W_1, ..., W_n)$, where $W_1, ..., W_n$ are distinct variables that also appear outside of any Skolem term in the same clause.

A CLP($\mathcal{BN}$) program in canonical form is a set of *clauses* of the form $H \leftarrow A/B$. We call H the head of the clause. H is a literal and A is a (possibly empty) conjunction of literals. Together they form the logical portion of the clause, C. The probabilistic portion, B, is a (possibly empty) conjunction of atoms of the form: $\{V = Sk \ \texttt{with} \ CPT\}$. We shall name these atoms *constraints*. Within a constraint, we refer to Sk as the Skolem term and CPT as the conditional probability table. We focus on discrete variables in this paper. In this case, CPT may be an unbound variable or a term or the form $\mathbf{p}(D, T, P)$. We refer to D as the domain, T as the table, and P as the parent nodes.

A CLP($\mathcal{BN}$) constraint B_i is well-formed if and only if:

1. all variables in B_i appear in C;
2. $Sk's$ functor is unique in the program; and,
3. there is at least one substitution σ such that $CPT\sigma = \mathbf{p}(D\sigma, T\sigma, P\sigma)$, and **(a)** $D\sigma$ is a ground list, all members of the list are different, and no subterm of a term in the list is a Skolem term; **(b)** $P\sigma$ is a ground list, all members of the list are different, and all members of the list are Skolem terms; and **(c)** $T\sigma$ is a ground list, all members of $T\sigma$ are numbers p such that $0 \leq p \leq 1$, and the size of $T\sigma$ is a multiple of the size of $D\sigma$.

If the probabilistic portion of a clause is empty, we also call the clause a *Prolog clause*. According to this definition, every Prolog program is a CLP($\mathcal{BN}$) program.

3.2 Operational Semantics

A query for CLP($\mathcal{BN}$) is an ordinary Prolog query, which is a conjunction of positive literals. In logic programming, a query is answered by one or more proofs constructed through resolution. At each resolution step, terms from two different clauses may be unified. If both of the terms being unified also participate in CPTs, or Bayes net constraints, then the corresponding nodes in the Bayes net constraints must be *unified* as illustrated in Figure 3. In this way we construct a large Bayes net consisting of all the smaller Bayes nets that have been unified during resolution.

A cycle may arise in the Bayes Net if we set a constraint such that Y is a parent of X, and X is already an ancestor of Y. In this case, when unifying Y to an argument of the CPT constraint for X, X would be a sub-term of the CPT

constraint for Y: we thus can use the occur-check test to guarantee the net is acyclic.

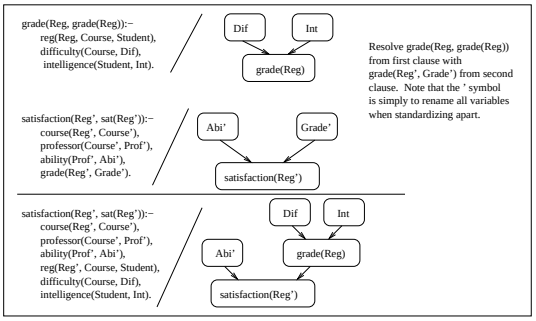


Figure 3: Resolution.

To be rigorous in our definition of the distribution defined by a Bayes net constraint, let C_i/B_i, $1 \leq i \leq n$, be the clauses participating in the proof, where C_i is the ordinary logical portion of the clause and B_i is the attached Bayes net, in which each node is labeled by a term. Let θ be the answer substitution, that is, the composition of the most general unifiers used in the proof. Note that during resolution a clause may be used more than once but its variables always are renamed, or standardised apart from variables used earlier. We take each such renamed clause used in the proof to be a distinct member of $\{C_i/B_i | 1 \leq i \leq n\}$. We define the application of a substitution θ to a Bayes net as follows. For each node in the Bayes net, we apply θ to the label of that node to get a new label. If some possible values for that node (according to its CPT) are not instances of that new label, then we marginalise away those values from the CPT.

3.3 Model-theoretic Semantics

A CLP($\mathcal{BN}$) program denotes a probability distribution over models. We begin by defining the probability distribution over ground Skolem terms that is specified by the probabilistic portion of a CLP($\mathcal{BN}$) program. We then specify the probability distribution over models, consistent with this probability distribution over ground Skolem terms, that the full CLP($\mathcal{BN}$) program denotes.

A CLP($\mathcal{BN}$) program P defines a unique joint probability distribution over ground Skolem terms as follows. Consider each ground Skolem term to be a random variable whose domain is a finite set of non-Skolem constants.[1] We now specify a Bayes net $\mathcal{BN}$ whose variables are these ground Skolem terms. Each ground Skolem term s is an instance of exactly one Skolem term t in the program P. To see this recall that, from the definition of Skolemization, any Skolem functor appears in only one term in the program P, and this one term appears in only one clause of P, though it my appear multiple times in that clause. Also from the definition of Skolemization, t has the form $sk(W_1, ..., W_n)$, where sk is a Skolem functor and $W_1, ..., W_n$ are distinct variables. Because s is a ground instance of t, $s = t\sigma$ for some substitution σ that grounds t. Because $t = sk(W_1, ..., W_n)$ appears in only one clause, t has exactly one associated (generalized) CPT, T, conditional on $W_1, ..., W_n$. Let the parents of s in $\mathcal{BN}$ be $W_1\sigma, ..., W_n\sigma$, and let the CPT be $T\sigma$. Note that for any node in $\mathcal{BN}$ its parents are subterms of that node. It follows that the graph structure is acyclic and hence that $\mathcal{BN}$ is a properly defined Bayes net, though possibly infinite. Therefore $\mathcal{BN}$ uniquely defines a joint distribution over ground Skolem terms; we take this to be the distribution over ground Skolem terms defined by the program P.

The meaning of an ordinary logic program typically is taken to be its least Herbrand model. Recall that the individuals in a Herbrand model are themselves ground terms, and every ground term denotes itself. Because we wish to consider cases where ground Skolem terms denote (non-Skolem) constants, we instead consider Herbrand quotient models [8]. In a Herbrand quotient model, the individuals are equivalence classes of ground terms, and any ground term denotes the equivalence class to which it belongs. Then two ground terms are equal according to the model if and only if they are in the same equivalence class. We take the set of minimal Herbrand quotient models for P to be those derived as follows.[2] Take the least Herbrand model of the logical portion of P, and for each non-Skolem constant, merge zero or more ground Skolem terms into an equivalence class with that constant. This equivalence class is a new individual, replacing the merged ground terms, and it participates in exactly the relations that at least one of its members participated in, in the same manner. It follows that each resulting model also is a model of P. The set of models that can be constructed in this way is the set S of minimal Herbrand quotient models of P. Let D be any probability distribution over S that is consistent with the distribution over ground Skolem terms defined by P. By consistent, we mean that for any ground Skolem term t and any constant c, the probability that $t = c$ according to the distribution defined by P is exactly the sum of the probabilities according to D of the models in which $t = c$. At least one such distribution D exists, since S contains one model for each possible combination of equivalences. We take such $\langle D, S \rangle$ pairs to be the models of P.

[1] This can be extended to a finite subset of the set of ground terms not containing Skolem symbols (functors or constants). We restrict ourselves to constants here merely to simplify the presentation.

[2] For brevity, we simply define these minimal Herbrand quotient models directly. In the full paper we show that it is sufficient to consider only Herbrand quotient models, rather than all logical models. We then define an ordering based on homomorphisms between models and prove that what we are calling the minimal models are indeed minimal with respect to this ordering.

3.4 Agreement Between Operational and Model-theoretic Semantics

Following ordinary logic programming terminology, the negation of a query is called the "goal," and is a clause in which every literal is negated. Given a program and a goal, the CLP($\mathcal{BN}$) operational semantics will yield a derivation of the empty clause if and only if every model $\langle D, S\rangle$ of the CLP($\mathcal{BN}$) program falsifies the goal and hence satisfies the query for some substitution to the variables in the query. This follows from the soundness and completeness of SLD-resolution. But in contrast to ordinary Prolog, the proof will be accompanied by a Bayes net whose nodes are labeled by Skolem terms appearing in the query or proof. The following theorem states that the answer to any query of this attached Bayes net will agree with the answer that would be obtained from the distribution D, or in other words, from the distribution over ground Skolem terms defined by the program P. Therefore the operational and model-theoretic semantics of CLP($\mathcal{BN}$) agree in a precise manner.

Theorem 1 *For any CLP($\mathcal{BN}$) program P, any derivation from that program, any grounding of the attached Bayes net, and any query to this ground Bayes net,*[3] *the answer to the query is the same as if it were asked of the joint distribution over ground Skolem terms defined by P.*

Proof: Assume there exists some program P, some derivation from P and associated ground Bayes net B, and some query $Pr(q|E)$ such that the answer from B is not the same as the answer from the full Bayes net $\mathcal{BN}$ defined by P. For every node in B the parents and CPTs are the same as for that same node in $\mathcal{BN}$. Therefore there must be some path through which evidence flows to q in $\mathcal{BN}$, such that evidence cannot flow through that path to q in B. But by Lemma 2, below, this is not possible.

Lemma 2 *Let B be any grounding of any Bayes net returned with any derivation from a CLP($\mathcal{BN}$) program P. For every query to B, the paths through which evidence can flow are the same in B and in the full Bayes net $\mathcal{BN}$ defined by P.*

Proof: Suppose there exists a path through which evidence can flow in $\mathcal{BN}$ but not in B. Consider the shortest such path; call the query node q and call the evidence node e. The path must reach q through either a parent of q or a child of q in $\mathcal{BN}$. Consider both cases. *Case 1*: the path goes through a parent p of q in $\mathcal{BN}$. Note that p is a parent of q in B as well. Whether evidence flows through p in a linear or diverging connection in $\mathcal{BN}$, p cannot itself have evidence—otherwise, evidence could not flow through p in $\mathcal{BN}$. Then the path from e to p is a shorter path through which evidence flows in $\mathcal{BN}$ but not B, contradicting our assumption of the shortest path. *Case 2*: the path from e to q flows through some child c of q in $\mathcal{BN}$. Evidence must flow through c in either a linear or converging connection. If a linear connection, then c must not have evidence; otherwise, evidence could not flow through c to q in a linear connection. Then the path from e to c is a shorter path through which evidence flows in $\mathcal{BN}$ but not B, again contradicting our assumption of the shortest path. Therefore, evidence must flow through c in a converging connection in $\mathcal{BN}$. Hence either c or one of its descendants in $\mathcal{BN}$ must have evidence; call this additional evidence node n. Since n has evidence in the query, it must appear in B. Therefore its parents appear in B, and their parents, up to q. Because evidence can reach c from e in B (otherwise, we contradict our shortest path assumption again), and a descendent of c in B (possibly c itself) has evidence, evidence can flow through c to q in B.

[3] For simplicity of presentation, we assume queries of the form $Pr(q|E)$ where q is one variable in the Bayes net and the evidence E specifies the values of zero or more other variables in the Bayes net.

4 Non-determinism and Aggregates

CLP($\mathcal{BN}$) can deal with aggregates with *no extra machinery*. As an example, in Fig 1 the rating of a course depends on the satisfaction of all students who registered. Assuming, for simplicity, that the rating is a deterministic function of the average satisfaction, we must average the satisfaction `Sat` for all students who have a registration `R` in course `C`. The next clause shows the corresponding program:

```
rating(C, Rat) :-
  setof(S,R^(registration(R,C),
            satisfaction(R,S)), Sats),
  average(Sats, CPT),
  {Rat = rating(C) with CPT}.
```

The call to `setof` obtains the satisfactions of all students registered in the course. The Prolog procedure `average` computes the conditional probability distribution of the course's rating as a function of student satisfaction. In our implementation, if the number students of small, the CPT will have manageable size. If the number of students grows large, using a single CPT is impractical. In this case `average` splits the CPT into a set of binary CPTs, which are merged together in a binary tree. The process, usually known as *divorcing*, is thus implemented automatically by the program.

5 Recursion

Recursion provides an elegant framework for encoding sequences of events. We next show a simple example of how to encode Hidden Markov Models.

The Brukian and Arinian have lived in a state of conflict for many years. The Brukian want to send their best spy, James Bound, to spy on the Arinian headquarters. The Arinian

watch commander is one of two men: Manissian is careful, but Lufy is a bit lax. The Brukian only know that if one of them was on watch today, it is likely they will be on watch tomorrow. The next program represents the information:

```
caught(0,Caught) :- !,
   {Caught = c(0) with p([t.f],[0.0,1.0],[])}.
caught(I,Caught) :-
   I1 is I-1, caught(I1, Caught0),
   watch(I, Police),
   caught(I,Caught0, Police, Caught).

watch(0, P) :- !,
   {P = p(0) with p([m,l],[0.5,0.5],[])}.
watch(I, P) :-
   I1 is I-1, watch(I1, P0),
   {P = p(I) with
       p([m,l],[0.8,0.2,0.8,0.2,0.8],[P0])}.

caught(I, C0, P, C) :-
   {C = c(I) with
      p([t,f],[1.0,1.0,0.05,0.001,
               0.0,0.0,0.95,0.099],[C0,P])}.
```

The variables `c(I)` give the probability James Bound was caught at or before time `I`, and `p(I)` gives the probabilities for who is watching at time `I`.

6 Relationship to PRMs

Clearly from the examples in Section 2, the CLP($\mathcal{BN}$) representation owes an intellectual debt to PRMs. As the reader might suspect at this point, any PRM can be represented as a CLP($\mathcal{BN}$) program. To show this, we next present an algorithm to convert any PRM into a CLP($\mathcal{BN}$) program. Because of space limits, we necessarily assume the reader already is familiar with the terminology of PRMs.

We begin by representing the skeleton of the PRM, i.e., the database itself with (possibly) missing values. For each relational table R of n fields, one field of which is the key, we define $n-1$ binary predicates $r_2, ..., r_n$. Without loss of generality, we assume the first field is the key. For each tuple or record $\langle t_1, ..., t_n \rangle$ our CLP($\mathcal{BN}$) program will contain the fact $r_i(t_1, t_i)$ for all $2 \leq i \leq n$. If t_i is a missing value in the database, then the corresponding fact in the CLP($\mathcal{BN}$) program is $r_i(t_1, sk_i)$, where sk_i is a unique constant that appears nowhere else in the CLP($\mathcal{BN}$) program; in other words, sk_i is a Skolem constant. It remains to represent the Bayes net structure over this skeleton and the CPTs for this structure.

For each field in the database, we construct a clause that represents the parents and the CPT for that field within the PRM. The head (consequent) of the clause has the form $r_i(\mathit{Key}, \mathit{Field})$, where the field is the i^{th} field of relational table R, and Key and $Field$ are variables. The body of the clause is constructed in three stages, discussed in the following three paragraphs: the relational stage, the aggregation stage, and the CPT stage.

The relational stage involves generating a translation into logic of each slot-chain leading to a parent of the given field within the PRM. Recall that each step in a slot chain takes us from the key field of a relational table R to another field, i, in that table, or vice-versa. Each such step is translated simply to the literal $r_i(X, Y)$, where X is a variable that represents the key of R and Y is a variable that represents field i of R, regardless of directionality. If the next step in the slot chain uses field i of table R, then we re-use the variable Y; if the next step instead uses the key of table R then we instead re-use variable X. Suppose field i is the foreign key of another table S, and the slot chain next takes us to field j of S. Then the slot chain is translated as $r_i(X, Y), s_j(Y, Z)$. We can use the same translation to move from field j of S to the key of R, although we would re-order the literals for efficiency. For example, suppose we are given a student key *StudentKey* and want to follow the slot chain through registration and course to find the teaching abilities of the student's professor(s). Assuming that the course key is the second field in the registration table and the student key is the third field, while the professor key is the second field of the course table, and ability is the second field of the professor table, the translation is as below. Note that we use the first literal to take us from *StudentKey* to *RegKey*, while we use the second literal to take us from *RegKey* to *CourseKey*.

$\mathit{registration}_3(\mathit{RegKey}, \mathit{StudentKey}),$
$\mathit{registration}_2(\mathit{RegKey}, \mathit{CourseKey}),$
$\mathit{course}_2(\mathit{CourseKey}, \mathit{ProfKey}),$
$\mathit{professor}_2(\mathit{ProfKey}, \mathit{Ability})$

In the preceding example, the variable *Ability* may take several different bindings. If this variable is a parent of a field, then the PRM will specify an aggregation function over this variable, such as *mean*. Any such aggregation function can be encoded in a CLP($\mathcal{BN}$) program by a predicate definition, as in ordinary logic programming, i.e. in Prolog. We can collect all bindings for *Ability* into a list using the Prolog built-in function *findall* or *bagof*, and then aggregate this list using the appropriate aggregation function such as *mean*. For the preceding example, we would use the following pair of literals to bind the variable X to the mean of the abilities of the student's professors.

$\mathit{findall}(\mathit{Ability}, \ (\ \mathit{registration}_2(\mathit{RegKey}, \mathit{CourseKey}),$
$\mathit{course}_2(\mathit{CourseKey}, \mathit{ProfKey}),$
$\mathit{professor}_2(\mathit{ProfKey}, \mathit{Ability}), \ L \ ),$
$\mathit{mean}(L, X)$

At this point, we have constructed a clause body that will compute binding for all the variables that correspond to parents of the field in question. It remains only to add a literal that encodes the CPT for this field given these parents.

The close link between PRMs and CLP($\mathcal{BN}$) raises the natural question, "given that we already have PRMs, of what utility is the CLP($\mathcal{BN}$) representation?" First, while there has been much work on incorporating probabilities into first-order logic, these representations have been far removed from the approach taken in PRMs. Hence while there is great interest in the relationship between PRMs and these various probabilistic logics, this relationship is difficult to characterise. Because CLP($\mathcal{BN}$)s are closely related to PRMs, they may help us better understand the relationship between PRMs and various probabilistic logics. Second, because CLP($\mathcal{BN}$) programs are an extension of logic programs, they permit recursion, non-determinism (a predicate may be defined by multiple clauses), and the use of function symbols, e.g., to construct data structures such as lists or trees. This expressivity may be useful for a variety of probabilistic applications. Of course we must note that the uses of recursion and recursive data structures are not unlimited. CLP($\mathcal{BN}$)s disallow resolution steps that introduce a cycle into a Bayes net constraint. Third, and most importantly from the authors' viewpoint, the following section of the paper demonstrates that the CLP($\mathcal{BN}$) representation is amenable to learning using techniques from inductive logic programming (ILP). Hence CLP($\mathcal{BN}$)s provides a way of studying the incorporation of probabilistic methods into ILP, and they may give insight into novel learning algorithms for PRMs. The methods of learning in PRMs [4] are based upon Bayes net structure learning algorithms and hence are very different from ILP algorithms. The CLP($\mathcal{BN}$) representation provides a bridge through which useful ideas from ILP might be transferred to PRMs.

7 Learning in CLP(BN) using ILP

This section describes the results of learning CLP(BN) programs using the ILP system ALEPH [11]. Other ILP systems could be applied in a similar manner.

7.1 The School Database

We have so far used the school database as a way to explain some basic concepts in CLP($\mathcal{BN}$), relating them to PRMs. The school database also provides a good example of how to learn CLP($\mathcal{BN}$) programs.

First, we use an interpreter to to generate a sample from the CLP($\mathcal{BN}$) program. The smallest database has 16 professors, 32 courses, 256 students and 882 registrations; the numbers roughly double in each successively larger database. We have no missing data. Can we, given this sample, relearn the original CLP($\mathcal{BN}$) program?

From the ILP point of view, this is an instance of multi-predicate learning. To simplify the problem we assume each predicate would be defined by a single clause. We use the Bayesian Information Criterion (BIC) score to compare alternative clauses for the same predicate. Because ALEPH learns clauses independently, cycles may appear in the resulting CLP($\mathcal{BN}$) program. We therefore augment ALEPH with a post-processing algorithm that simplifies clauses until no cycles remain; the algorithm is greedy, choosing at each step the simplification that will least affect the BIC score of the entire program.

The following is one of the learned CLP($\mathcal{BN}$) clauses; to conserve space, we simply write "CPT" instead of showing the large table.

```
registration_grade(A,B) :-
    registration(A,C,D), course(C,E),
    course_difficulty(C,F), student_intelligence(D,G),
    {F = registration_grade(A) with p([a,b,c,d],CPT,[F,G]}.
```

Figure 4 illustrates, as a PRM-style graph, the full set of clauses learned for the largest of the databases before simplification; this would be the best network according to BIC, if not for the cycles. Figure 5 plots various natural measures of the match between the learned program *after cycles have been removed* and the original program, as the size of the database increases. By the time we get to the largest of the databases, the only measures of match that do not have a perfect score are those that deal with the directions of arcs.

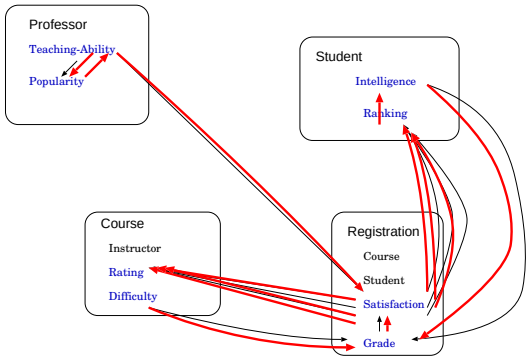


Figure 4: Pictorial representation of the CLP($\mathcal{BN}$) clauses learned from the largest schools database, before removal of cycles.

7.2 Metabolic Activity

Our second application is based on data provided for the 2001 KDD Cup. We use the KDD01 Task 2 training data, consisting of 4346 entries on the activities of 862 genes [3]. We chose to study this problem because it is a real-world relational problem with much missing data. We concentrate on the two-class problem of predicting whether a gene codes for metabolism, because it illustrates a strength of learning a CLP($\mathcal{BN}$) program rather than an ordinary logic program.

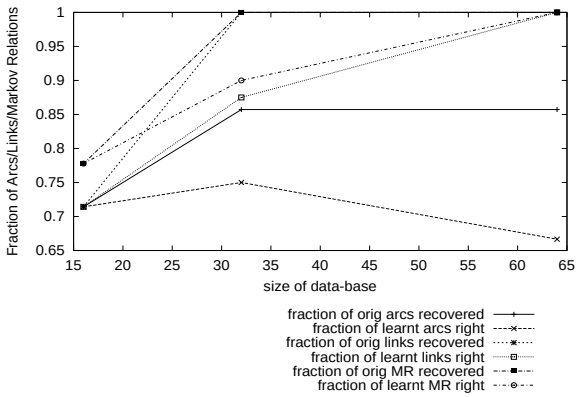


Figure 5: Graph of results of CLP($\mathcal{BN}$)-learning on the three sizes of schools databases. Links are arcs with direction ignored. A Markov relation (MR) holds between two nodes if one is in the Markov blanket of the other.

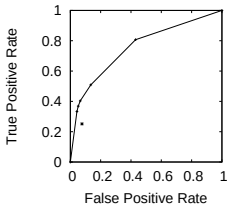


Figure 6: ROC Curve for Metabolism Application

A theory predicts metabolism if we have at least a proof saying that the probability for metabolism is above a certain value T^M, and no proof saying that the probability for metabolism is below a certain T_m. We allow T^M to range from 0.3 to 0.9. Because we obtain probabilities, it is straightforward to generate an ROC curve showing how we can gain a higher true positive prediction rate if we are willing to live with a higher false positive prediction rate. This is an advantage of learning a CLP($\mathcal{BN}$) program rather than an ordinary logic program. The ROC curve is shown in Figure 6. The performance of the ordinary logic program learned by ALEPH is a single point, lying slightly (not significantly) below the curve shown.

8 Conclusion

This paper has defined CLP($\mathcal{BN}$), an extension of logic programming that defines a joint probability distribution over the denotations of ground Skolem terms. Because such terms are used in logic programming in place of existentially-quantified variables, or missing values, CLP($\mathcal{BN}$) is most closely related to PRMs. The full paper also gives a detailed discussion of the relationship of CLP($\mathcal{BN}$) to other probabilistic logics, including the work of Breese [2], Haddawy and Ngo [5], Sato [10], Poole [9], Koller and Pfeffer [7], Angelopolous [1], and Kersting and DeRaedt [6]. To summarise that discussion in a sentence, CLP($\mathcal{BN}$) does not replicate any of these approaches because they define probability distributions over sets of objects other than the set of ground Skolem terms.

CLP($\mathcal{BN}$) is a natural extension of logic programming that subsumes PRMs. While this does not imply that CLP($\mathcal{BN}$) is preferable to PRMs, its distinctions from PRMs follow from properties of logic programming. These distinctions include the combination of function symbols and recursion, non-determinism (a predicate may be defined by multiple clauses), and the applicability of ILP techniques for learning CLP($\mathcal{BN}$) programs. Our primary direction for further work is in developing improved ILP algorithms for learning CLP($\mathcal{BN}$) programs.

References

[1] N. Angelopoulos. *Probabilistic Finite Domains*. PhD thesis, Dept of CS, City University, London, 2001.

[2] J. S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 8(4):624–647, 1992.

[3] J. Cheng, C. Hatzis, H. Hayashi, M.-A. Krogel, S. Morishita, D. Page, and J. Sese. KDD Cup 2001 report. *SIGKDD Explorations*, 3(2):47–64, 2002.

[4] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, chapter 13, pages 307–335. Springer, Berlin, 2001.

[5] P. Haddawy. An overview of some recent developments in Bayesian problem solving techniques. *AI Magazine*, Spring 1999.

[6] K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Germany, April 2001.

[7] D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI-97*, Nagoya, Japan, August 1997.

[8] C. D. Page. *Anti-unification in constraint logics*. PhD thesis, University of Illinois at Urbana-Champaign, 1993. UIUCDCS-R-93-1820.

[9] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[10] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[11] A. Srinivasan. *The Aleph Manual*, 2001.

Appears in the *Proc. of the Intl. Conf. on Inductive Logic Programming, July 2002, Sydney, Australia.*

An Empirical Evaluation of Bagging in Inductive Logic Programming

Inês de Castro Dutra, David Page, Vítor Santos Costa and Jude Shavlik

Department of Biostatistics and Medical Informatics and
Department of Computer Sciences,
University of Wisconsin-Madison, USA

Abstract. Ensembles have proven useful for a variety of applications, with a variety of machine learning approaches. While Quinlan has applied boosting to FOIL, the widely-used approach of bagging has never been employed in ILP. Bagging has the advantage over boosting that the different members of the ensemble can be learned and used in parallel. This advantage is especially important for ILP where run-times often are high. We evaluate bagging on three different application domains using the complete-search ILP system, Aleph. We contrast bagging with an approach where we take advantage of the non-determinism in ILP search, by simply allowing Aleph to run multiple times, each time choosing "seed" examples at random.

1 Introduction

Inductive Logic Programming (ILP) systems have been quite successful in extracting comprehensible models of relational data. Indeed, for over a decade, ILP systems have been used to construct predictive models for data drawn from diverse domains. These include the sciences [16], engineering [10], language processing [33], environment monitoring [11], and software analysis [5]. In a nutshell, ILP systems repeatedly examine candidate clauses (the "search space") to find good rules. Ideally, the search will stop when the rules cover nearly all positive examples with only a few negative examples being covered.

Unfortunately, the search space can grow very quickly in ILP applications. Several techniques have therefore been proposed to improve search efficiency. Such techniques include improving computation times at individual nodes [4, 26], better representations of the search [3], sampling the search space [27, 28, 32], and parallelism [8, 13, 19]. Parallelism can be obtained from very different alternative approaches, such as dividing the search tree, dividing the examples, or even through performing cross-validation in parallel [31].

An intriguing alternative approach that can lead to better accuracy whilst taking advantage of parallelism is the use of *ensembles*. Ensembles are classifiers

that combine the predictions of multiple classifiers to produce a single prediction [9]. To some extent, an induced theory is an ensemble of clauses. We would like to go one step further and *combine different theories to form a single ensemble*. The main advantage is that the ensemble is often more accurate than its individual components. Moreover, we can use parallelism both in generating and in actually evaluating the ensemble.

Several methods have been presented for ensemble generation. In this work, we concentrate on a popular method that is known to generally create a more accurate ensemble than individual components, *bagging* [6]. Bagging works by training each classifier on a random sample from the training set. In contrast to other well-known techniques for ensemble generation, such as boosting [12], bagging has the important advantage that it is effective on "unstable learning algorithms" [7], where small variations in parameters can cause huge variations in the learned theories. This is the case with ILP. A second advantage is that it can be implemented in parallel trivially.

We contrast bagging with a method we name *different seeds*, where we try to take advantage of the non-determinism in seed-based search by simply combining different theories obtained from experimenting with different seed examples, while always using the original training set. This method is also easily parallelisable.

Several researchers have been interested in the use of ensemble-based techniques for Inductive Logic Programming. To our knowledge, the original work in this area is Quinlan's work on the use of boosting in FOIL [25]. His results suggested that boosting could be beneficial for first-order induction. More recently, Hoche proposed confidence-rated boosting for ILP with good results [15]. Zemke recently proposed bagging as a method for combining ILP classifiers with other classifiers [34]. The contributions of our paper are to experimentally evaluate bagging on three particularly challenging ILP applications, and to compare bagging with the approach of different seeds.

The paper is organised as follows. First, we present in more detail ensemble techniques, focusing on bagging. Next, we discuss our experimental setup and the applications used in our study. We then discuss how ensembles perform on our benchmarks. Last, we offer our conclusions and suggest future work.

2 Ensembles

Ensembles aim at improving accuracy through combining the predictions of multiple classifiers in order to obtain a single classifier. Experience has shown that ensemble-based techniques such as bagging and boosting can be very effective for decision trees and neural networks [24, 21]. On the other hand, there has been less empirical testing with classifiers as logic programs.

Figure 1 shows the structure of an ensemble of logic programs. This structure can also be used for classifiers other than logic programs. In the figure, each program $P_1, P_2, \ldots, P_N$ is trained using a set of training instances. At classification time each program receives the same input and executes on it independently.

The outputs of each program are then combined and an output classification reached. Figure 1 illustrates that in order to obtain good classifiers one must address three different problems:

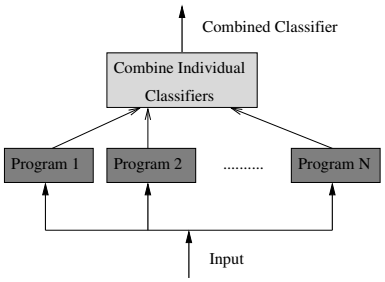


Fig. 1. An Ensemble of Classifiers.

- how to generate the individual programs;
- how many individual programs to generate;
- how to combine their outputs.

Regarding the first problem, research has demonstrated that a good ensemble is one where the individual classifiers are accurate and make their errors in different parts of the instance space [17, 22]. Obviously, the output of several classifiers is useful only if there is disagreement between them. Hansen and Salamon [14] proved that if the average error rate is below 50% and if the component classifiers are independent, the combined error rate can be reduced to 0 as the number of classifiers goes to infinity.

Methods for creating the individual classifiers therefore focus on producing classifiers with some degree of diversity. In the present work, we follow two approaches to producing such classifiers, described in the two following paragraphs.

One interesting aspect of Inductive Logic Programming is that the same learning algorithm may lead to quite different theories. More specifically, theories generated by seed-based ILP algorithms may heavily depend on the choice of the seed example. A natural approach to generate ensembles is to take advantage of this property of ILP systems, and combine the rather different theories that were generated just by choosing different sequences of seed examples. We call this approach *different seeds*.

We contrast the random choice of seeds with bagging. Bagging classifiers are obtained by training each classifier on a random sample of the training set. Each classifier's training set is generated by randomly, uniformly drawing K examples with *replacement*, where K is the size of the original training set. Thus, many of the original examples may be repeated in the classifier's training set.

Table 1. Example of Bagging Training Sets.

Training Sets	Examples Included					
1	6	2	6	3	2	5
2	1	4	6	5	1	6
3	1	3	3	5	2	3
4	6	4	1	4	3	2
5	6	4	2	3	2	3
6	5	5	2	1	5	4

Table 1 shows six training sets randomly generated from a set with examples numbered from 1 to 6. We can notice that each bagging training set tends to focus on different examples. The first training set will have two instances of the second and sixth examples, while having no instances of the second and fourth example. The second example will have instead two occurrences of the first and sixth example, while missing the second and third example. In general, accuracy for each individual bagging classifier is likely to be lower than for a classifier trained on the original data. However, when combined, bagging classifiers can produce accuracies higher than that of a single classifier, because the diversity among these classifiers generally compensates for the increase in error rate of any individual classifier.

Therefore, the promise of bagging, and of ensembles in general, is that when classifiers are combined the accuracy will be higher than the accuracy for the original classifier. In our case, one particularly interesting result for bagging is that it is effective on "unstable" learning algorithms, that is, on algorithms where a small change in the training set may lead to large changes in prediction. We focus on bagging in this work because it is considered to be less vulnerable to noise than boosting algorithms [12] and it can take advantage of parallel execution.

The second issue we had to address was the choice of how many individual classifiers to combine. Previous research has shown that most of the reduction in error for ensemble methods occurs with the first few additional classifiers [14]. Larger ensemble sizes have been proposed for decision trees, where gains have been seen up to 25 classifiers. In our experiments we decided to extend our analysis up to 100 classifiers.

The last problem concerns the combination algorithm. An effective combining scheme is often to simply average the predictions of the network [1, 7, 17, 18]. An alternate approach relies on a pre-defined *voting threshold*. If the number of theories that cover an example is above or equal to the threshold, we say that the example is positive, otherwise the example is negative. Thresholds may range from 1 to the ensemble size. A voting threshold of 1 corresponds to a classifier that is the disjunction of all theories. A voting threshold equal to the ensemble size corresponds to a classifier that is the conjunction of all theories.

3 Methodology

We use the ILP system Aleph [29] in our study. Aleph assumes (a) background knowledge B in the form of a Prolog program; (b) some language specification $\mathcal{L}$ describing the hypotheses; (c) an optional set of constraints I on acceptable hypotheses; and (d) a finite set of examples E. E is the union of a nonempty set of "positive" examples E^+, such that none of the E^+ are derivable from B, and a set of "negative"examples E^-.

Aleph tries to find one hypothesis H in $\mathcal{L}$, such that: (1) H respects the constraints I; (2) The E^+ are derivable from B, H, and (3) The E^- are not derivable from B, H. By default, Aleph uses a simple greedy set cover procedure that constructs such a hypothesis one clause at a time. In the search for any single clause, Aleph selects the first uncovered positive example as the seed example, saturates this example, and performs an admissible search over the space of clauses that subsume this saturation, subject to a user-specified clause length bound.

We have elected to perform a detailed study on three datasets, corresponding to three non-trivial ILP applications. For each application we ran Aleph with random re-ordering of the positive examples and hence of seeds. We call this experiment *different seeds*. Next, we created bagged training sets from the original set, and called Aleph once for each training set. We call this experiment *bagging*. The number of runs of *different seeds* is the same as the number of bags.

Aleph allows the user to set a number of parameters. We always set the following parameters as follows:

- search strategy: `search`. We set it to be breadth-first search, `bf`. This enumerates shorter clauses before longer ones. At a given clauselength, clauses are re-ordered based on their evaluation. This is the Aleph default strategy that favours shorter clauses to avoid the complexity of refining larger clauses.
- evaluation function: `evalfn`. We set this to be coverage. Clause utility is measured as $P-N$, with P and N being the number of positive and negative examples covered by the clause, respectively.
- chaining of variables: `i`. This Aleph parameter controls variable chaining during saturation: chaining depth of a variable that appears for the first time in a literal $\mathcal{L}_i$, is 1 plus the maximum chaining depth of all variables that appear in previous literals $\mathcal{L}_j, j < i$. We used a value of 5 instead of the default value of 2 in order to obtain more complex relations between literals in a clause.
- max number of nodes allowed: `maxnodes`. This corresponds to the number of clauses in the search space. We set this to be 100,000.
- maximum number of literals in a clause: `maxclauselength`. This was chosen to be the largest clause length that produced run times smaller than 1 hour on Intel 700 MHz machines, running Linux Red Hat 6.2. For our applications this value was either 4 or 5.

For each application, we ran experiments with different lower bounds on the minimum accuracy of an acceptable clause (`minacc`). We chose the values of 0.7,

0.9 and 1.0. In order to keep running times feasible, we first ran our datasets at least 5 times with the three different minaccs, and also with clause length varying from 4 to 10. We then chose the parameters that allowed Aleph to run at most for one hour (on Intel 700 MHz machines). It happened that for all minaccs, the clause length that produced runtimes less than or equal to one hour was the same, though it varied from one application to another. For each application we thus will vary our `minacc` settings among 0.7, 0.9 and 1.0, and we use the appropriate `maxclauselength`.

Next, we organise our discussion of methodology into (a) experimentation and (b) evaluation.

Experimentation. Our experimental methodology employs five-fold cross-validation. For each fold, we consider ensembles with size varying from 1 to 100. The *minacc* parameter used to generate the component theories in an ensemble, and the voting threshold used for the ensembles, are tunable parameters. These parameters are tuned within each fold, using only the training data for that fold. Moreover, rather than holding out a single tuning set from the training data for a fold, we perform 4-fold tuning within the training set. The parameter combination that yields the highest accuracy during this "inner" 4-fold cross-validation on the training set is then used on the entire training set for that fold. The resulting theory is then tested on the test set for that fold. The process is repeated for each of the five folds, and the results are merged in the standard way for cross-validation.

Next, we present the details of the experimental setup, starting with tuning. As explained, our goal in the tuning phase is to estimate what is the best parameter setting (minacc, voting threshold) for the ensembles, in order to use them later during the training/test phase. Tuning proceeds in two steps. First, we repeatedly call Aleph to generate all the theories we need to construct the different ensembles. Because the ILP runs are time-consuming, we do not repeat the ILP runs themselves to learn entirely new theories for each different ensemble size. Rather, for each tuning fold and minacc parameter, we initially learn 100 theories using *different seeds* methodology and 100 theories using *bagging*. Then, for either ensemble approach, and for each ensemble size s between 1 and 99 we randomly select s different theories from the 100. We next use these theories to generate ensembles, and evaluate the results. Because our results may be distorted by a particularly poor or good choice of these theories, we repeat this selection process 30 times and average the results.

Tuning thus requires 12,000 theories per application: one theory for *bagging* plus one theory for *different seeds*, times 5 test set folds, times 4 tuning folds, times 3 minacc values, times the 100 different theories we create for producing ensembles.

Once the 12,000 theories are generated, two tables are produced. The first one, *minaccs_for_rocs*, is used to calculate the ROC curves and contains the best minacc for each ensemble size and for each voting threshold. The second one, *best_thresholds* is used to obtain the accuracies and contains the best combination

of minacc and voting threshold for each ensemble size. This second table is a subset of *minaccs_for_rocs*.

We next move to the 5-fold cross-validation by doing a training/test per fold, per each minacc. First, we produce 600 theories per fold: one for bagging plus one for *different seeds*, times 3 minaccs, times the 100 different theories used to create ensembles. We are now ready to evaluate the ensembles.

Evaluation. For the evaluation phase (b), we used two techniques to evaluate the quality of the ensembles. First, we studied how average accuracy varies with ensemble size. We present accuracy as the average between accuracy on the positive examples and accuracy on the negative examples.

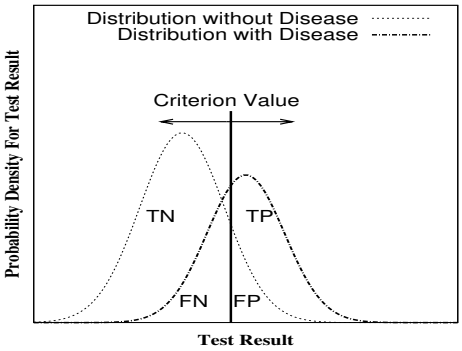


Fig. 2. Example of Probability Density Functions for Two Populations: with Disease, without Disease.

Second, we studied Receiver Operating Characteristic (ROC) curves for the ensembles. Provost and Fawcett [23] have shown how ROC curve analysis [20, 35] can be used to assess classifier quality. When we consider the results of a particular test in two populations, say positive and negative examples, we will rarely observe a perfect separation between the two groups. Indeed, the distribution of the test results can overlap, as shown in Figure 2. For every possible cut-off point or criterion value we select to discriminate between two populations, there will be some cases with the classifier correctly reporting positive examples to be positive (TP = True Positive fraction), but some cases incorrectly reported negative (FN = False Negative fraction). On the other hand, some negative examples will be correctly classified as negative (TN = True Negative fraction), but some negative examples will be classified as positive (FP = False Positive fraction).

In an ROC curve the true positive rate (sensitivity, or $\frac{TP}{TP+FN}$) is plotted against the false positive rate (100-specificity, or $\frac{FP}{FP+TN}$) for different cut-off points. Each point on the ROC plot represents a sensitivity/specificity pair corresponding to a particular decision threshold. A test with perfect discrimination (no overlap in the two distributions) has a ROC plot that passes through the

upper left corner (100% sensitivity, 100% specificity). Therefore the closer the ROC plot is to the upper left corner, the higher the overall accuracy of the test [35].

When we have many ROC curves to be analysed, we can look instead to the area under those curves. The value for the area under the ROC curve can be interpreted as follows: an area of 0.84, for example, means that a randomly selected individual from the positive group has a test value larger than that for a randomly chosen individual from the negative group 84% of the time. When the variable under study cannot distinguish between the two groups, i.e. where there is no difference between the two distributions, the area will be equal to 0.5 (as is the case when the ROC curve coincides with the diagonal). When there is a perfect separation of the values of the two groups, i.e. there is no overlapping of the distributions, the area under the ROC curve equals 1 (the ROC curve will reach the upper left corner of the plot).

We wish to test the effectiveness of different sizes of ensembles, from 1 to 99. Again, we do not repeat the ILP runs themselves to learn entirely new theories for each different ensemble size. Rather, we use the theories from the previous step. Because our results may be distorted by a particularly poor or good choice of these theories, we repeat this selection process 30 times and average the results.

Accuracies across the folds are obtained by averaging the sum of all positives and negatives for every fold. Areas across the folds are obtained by simply averaging areas computed for each ensemble size. ROC curves across the folds are obtained by averaging the rates of positives and negatives of each fold.

All experiments were performed using Condor, a tool for managing heterogeneous resources developed by the Condor Team at the UW-Madison [2]. Without the utilisation of such a tool, our experiments would have taken years to be concluded. Our jobs occupied about 53,380 hours of CPU, with an average peak of 400 jobs running simultaneously on Intel/Linux and Sun4u/Solaris machines.

3.1 Benchmark Datasets

Our benchmark set is composed of three datasets that correspond to three non-trivial ILP applications. We next describe the characteristics of each dataset with its associated ILP application, and present a dataset summary table.

Carcinogenesis. Our first application concerns the prediction of carcinogenicity test outcomes on rodents [30]. This application has a number of attractive features: it is an important practical problem; the background knowledge consists of large numbers of non-determinate predicate definitions; experience suggests that a fairly large search space needs to be examined to obtain a good clause.

Smuggling. Our second dataset concerns data on smuggling of some materials. The key element of our data is a set of smuggling *events*. Different events may be related in a variety of ways. They may share a common location, they may involve the same materials, or the same person may participate. Detailed data on people,

locations, organisations, and occupations is available. The actual database has over 40 relational tables. The number of tuples in a relational table vary from 800 to as little as 2 or 3 elements.

The ILP system had to learn which events were *related*. We were provided with a set of related examples that we can use as positive examples. We can assume all other events are unrelated and therefore compose a set of negative examples. We assume *related* is comutative. Therefore we changed Aleph to assume `related(B,A)` if `related(A,B)` was proven, and vice-versa.

The smuggling problem is thus quite challenging in that it is a heavily relational learning problem over a large number of relations, whereas most traditional ILP applications usually require a small number of relations.

Protein. Our last dataset consists of a database of genes and features of the genes or of the proteins for which they code, together with information about which proteins interact with one another and correlations among gene expression patterns. This dataset is taken from the function prediction task of KDD Cup 2001. While the KDD Cup task involved 14 different protein functions, our learning task focuses on the challenging function of "metabolism": predicting which genes code for proteins involved in metabolism. This is not a trivial task for our ILP system.

Table 2. Datasets Characteristics.

	Dataset Sizes	Max Clause Length
Carcinogenesis	182+/148-	4
Smuggling	143+/517-	5
Protein	172+/690-	5

Table 2 summarises the main characteristics of each application. The second column corresponds to the size of the full datasets, where `P+/N-` represents number of positive examples and number of negative examples. Bags are created by randomly picking elements, with replacement, from the full dataset. Therefore the number of positives or negatives of each bag are not the same as of the full dataset used for *different seeds*, although the total size is the same. The second column indicates the clause length used for each dataset. The test sets for each 5-fold cross-validation experiment is obtained by a block distribution of the full dataset. For example, application Carcinogenesis will have 5 positive test sets of sizes: 36, 36, 36, 36 and 38. These test sets are not used during the tuning phase.

4 Results

This section presents our results and analyses the performance of each application. For each application we show the average accuracy for positives and

negatives, and the area under ROC curves built from 1 to 25 ensemble sizes. The results from 26 to 99 are essentially horizontal lines and are not shown. We report results for *different seeds* and *bagging*. We also show the ROC curve for an ensemble size of 25.

The accuracies presented in the graphs are averaged across all folds, and for each ensemble size, a different voting threshold and minacc are used. This combination of voting threshold and minacc is the one that produced the best accuracy during the tuning phase. For clarity's sake, these parameter values are not shown in the curves.

The areas under the ROC curves were computed by (1) computing an ROC curve, per fold, for each ensemble size using the theories learned for the best pair *<minacc, voting threshold>*, (2) computing the area under each ROC curve, and (3) averaging the areas for each ensemble size across the folds.

Figure 3 shows the average accuracies for the three applications, for *different seeds* and *bagging*, when varying the ensemble sizes. Figure 4 shows the areas under the ROC curves, averaged across five folds, for the three applications, when varying the ensemble sizes. Figure 5 shows ROC curves at ensemble size 25 for every application.

The results show that ensembles do provide an improvement both in accuracies and in ROC areas. Most of the improvement is obtained for smaller ensemble sizes, up to 5 or 10 elements. Performance does not seem to benefit much from using larger sizes.

The best results were obtained in the smuggling application, with *bagging* and *different seeds* obtaining similar performance. The most irregular application is carcinogenesis. We discuss the individual applications in more detail in the next sections.

4.1 Carcinogenesis

Single-theory accuracy results for carcinogenesis are not very good. On average, accuracy for a single theory is around 60% for *different seeds* and 59% for *bagging*. Our results show that ensembles can improve accuracy to around 64%. Unfortunately, our results also show huge variations, as we discuss next.

Figure 3(a) shows the average among the accuracy curves for the five folds. At first, the curves show that both *bagging* and *different seeds* obtain significant improvements. As ensemble size grows, *different seeds* tends to obtain better results, whereas *bagging* on average tends to achieve performance closer to the single-theory case. Both curves show a number of peaks.

The maximum average accuracy for *different seeds* is 64.5%, which represents a significant improvement over the single-theory accuracy. Studying *different seeds* in more detail, we found the system tends to be pretty reasonable at classifying positive examples. It achieves a maximum of 78.7% acceptance rate for *different seeds*, at ensemble size 31. It does not perform so well at rejecting negative examples: the best result is a minimum probability of 48.3% of rejecting a negative example, at ensemble size 12.

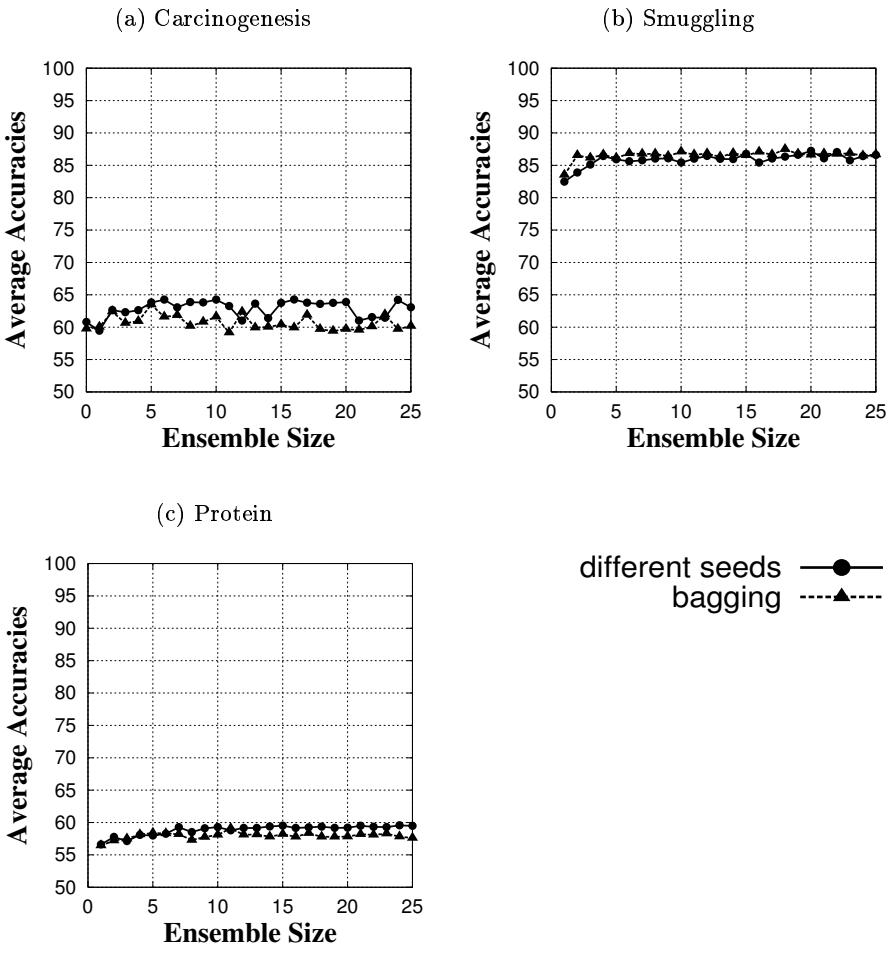


Fig. 3. Average Accuracies for the Three Applications.

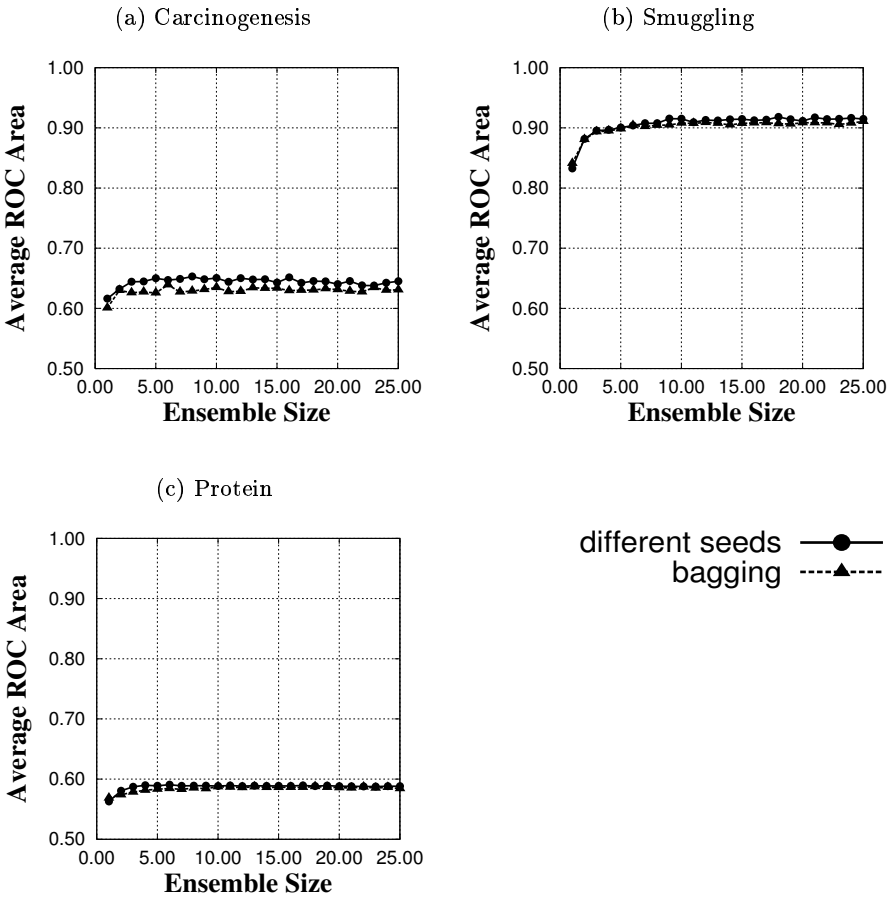


Fig. 4. Areas under the ROC curves for the Three Applications.

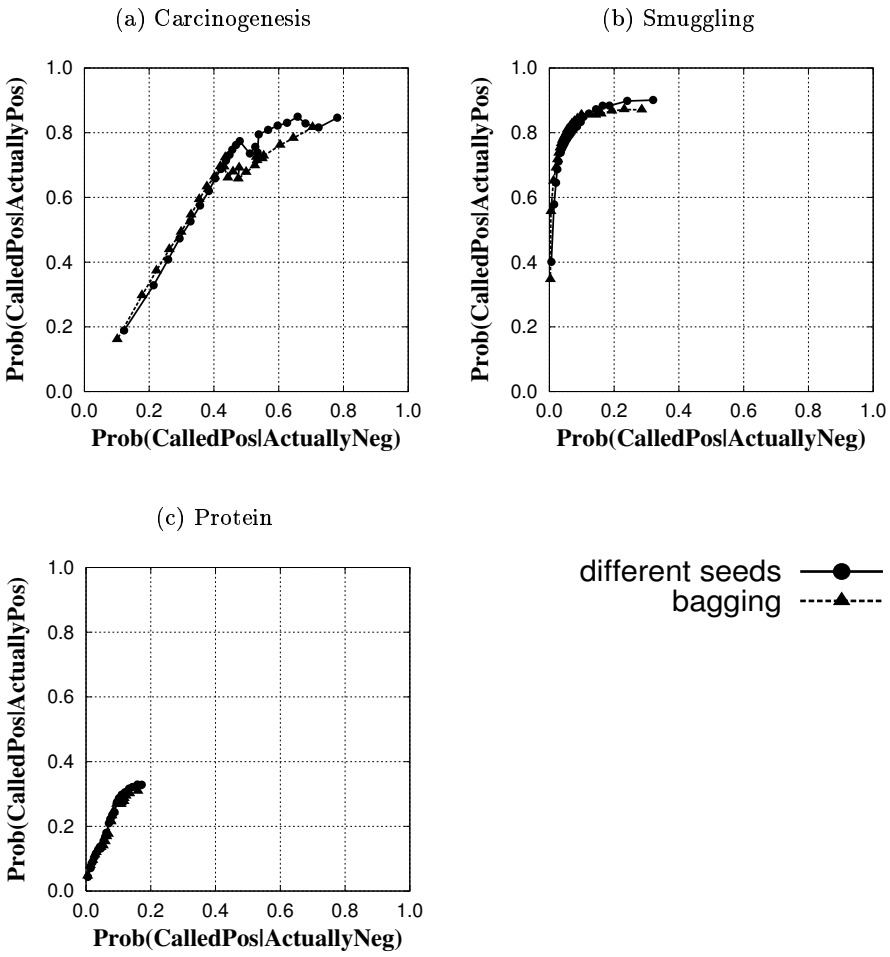


Fig. 5. ROC curves at N=25 for the Three Applications.

Different seeds is particularly interesting in that it shows several spikes, which are most obvious in the accuracy curve. This effect is caused by our tuning algorithm that has some difficulties with a very unstable application, such as carcinogenesis. More precisely, study of the data shows that often the tuning algorithm will choose the same minacc in a row for a series of consecutive ensemble sizes in a fold, and then all of a sudden select a different minac, and immediately return to choosing the initial parameter. The points where there is an abrupt change of parameters are usually the points where we have spikes.

The accuracy curve for *bagging* has several spikes on smaller ensembles, and then stabilises rather quickly. Again, most of the variation is caused by the choice of parameters. Interestingly enough, whereas most spikes in *different seeds* are negative, in this case most spikes are improvements. This suggests that accuracy results may be suffering from a wrong choice, either of thresholds or of Aleph's minimal training accuracy parameter.

Next, we look at the areas under the ROC curves as ensemble size varies from size 1 to 25. The area under a ROC curve gives a good estimate of classifier quality.

Figure 4(a) compares the ROC areas for *different seeds* and for *bagging*. We can notice that both techniques obtain a significant improvement with the smaller ensembles. *Bagging* stabilises very quickly, though, whereas *different seeds* obtains improvements up to ensemble size 5. As a result, *different seeds* has a somewhat better result than *bagging*.

ROC curves provide a more detailed picture of the behaviour of this application concerning rates of true positives against false positives. We chose to plot an ROC curve for ensemble size of 25, the largest ensemble size for which we present other results. Figure 5(a) shows an ROC curve for ensemble size 25, for the application Carcinogenesis, for *different seeds* and *bagging*, when varying the voting threshold from 1 to 25. The curves show very clearly the spikes caused by the different minacc chosen for each point.

The curves show large variations for small ensemble sizes. This corresponds to choosing different minaccs. Both *different seeds* and *bagging* can achieve a very good improvement on positive examples. However, at a cost of increasing the rate of false positives. The best result for positives, for *different seeds* is achieved at voting threshold 1, with acceptance rate of 88%. *Bagging* also achieves its best performance on positives at threshold 1, but it performs a little worse than *different seeds* achieving maximum of 85% of acceptance rate. As the voting threshold increases, the chance of better classifying a positive or negative example decreases. Note that the voting threshold increases as we decrease along the X axis, but not in a consecutive order of points in the ROC curve. For example, at ensemble size 9, the false positive rate is 0.66, and the true positive rate is 0.81, while at ensemble size 10, the false positive rate is 0.70 and the true positive rate is 0.80.

Our main conclusion is that *different seeds* performs better for small thresholds, that is, it can recognise most positive examples. *Bagging* tends to perform better for large thresholds, that is, it is better at recognising negative examples.

4.2 Smuggling

The smuggling problem is quite challenging in that it is a heavily relational learning problem over a large number of relations, whereas most traditional ILP applications usually require a small number of relations. It was therefore quite interesting to find that Aleph can achieve quite good accuracy for this problem. We found average accuracy to be about 82% for *different seeds* and 83% for *bagging*, for a single theory. Single accuracy for negative examples for a single theory is around 89% for *different seeds* and 91% for *bagging*, while accuracy for positive examples are around 76%, for both *different seeds* and *bagging*.

Of course, it would be quite nice to achieve an even better accuracy. Figure 3(b) shows the variation of accuracy averaged across folds, as we range the size of the ensembles between 1 and 25. Accuracy for both curves increases quickly for smaller ensembles and then is largely stable as we increase ensemble size. *Different seeds* and *bagging* achieve a maximum average accuracy of around 87%, with *different seeds* behaving slightly better than *bagging* for larger ensemble sizes. Our results thus correspond to a significant improvement over the single-theory accuracy. Accuracies stabilise at around 92%, for both *different seeds* and *bagging*, at classifying negative examples, and at around 82% for both *different seeds* and *bagging*, at classifying positive examples.

This application illustrates the benefit of using *bagging* at smaller ensemble sizes, and stresses the advantage of using ensemble methods to improve the accuracy of a single theory.

Figure 4(b) shows the areas under the ROC curves from ensemble size 1 to 25 averaged across the five test set folds. The results on ROC areas are very impressive. Performance is excellent for both *different seeds* and *bagging*, with a small advantage for *different seeds*. Both curves show a substantial improvement up to size 10, and then stabilise. Also notice that *bagging* and *different seeds* achieve very similar results.

Figure 5(b) shows average of ROC points across five folds, for ensemble size 25 varying the threshold from 1 to 25. Both classifiers perform in much the same way. The combined classifier is very good at classifying negative examples: even in the worst case, it only misclassifies up to 30% of all negative examples. Results are also quite good on the positive examples, although some examples are never covered. *Different seeds* does have some advantage in this case. Last, notice that there are much less spikes than for Carcinogenesis: the tuning algorithm seems to perform quite well here.

4.3 Protein

The Protein application has average single-theory accuracy of around 56% for both *different seeds* and *bagging*. Figure 3(c) shows the average accuracies between positives and negatives for *different seeds* and *bagging*, as we increase the ensemble size from 1 to 25. *Bagging* and *different seeds* have very similar performance for this application. The improvement over the single-theory is between 2

and 3 percentage points, and does show that the ensemble methods can improve performance even in this case.

Different seeds achieves maximum average accuracy of 60% at ensemble size 47, while *bagging* achieves maximum accuracy of 59% at ensemble size 28.

In order to understand better what happens with this application we draw the ROC curve for ensemble size 25. Figure 5(c) shows the ROC curves for *different seeds* and *bagging* averaged across five folds. The performance of this application on positives improves as we increase the ensemble size for both *different seeds* and *bagging*. For low thresholds, *different seeds* does better for positives. The learned theories seem to have difficulty in generalising: we cannot cover most examples. This results in bad accuracy for positives, and in good accuracy for negatives. The results also show that most of the improvement happens for smaller ensembles.

Our analysis provides more insight when we look at the areas under the ROC curves, varying our threshold from 1 to 25. Figure 4(c) shows the areas under the ROC curves. The results essentially confirm what we obtained with accuracy. *Different seeds* and *bagging* have the same performance up to ensemble size 24. After that *different seeds* surpasses the performance of *bagging*.

As with the other applications, both ensemble methods improve performance over the single-theory.

5 Conclusions and Future Work

This work presents an empirical study of *bagging*, a well-known ensemble building mechanism, in the Inductive Logic Programming setting. In our approach, we use bagging to combine theories built from random variations of the original training set. We contrast bagging to *different seeds*, an approach where we always use the same training set, and randomly select different seeds to build the different theories in the ensemble. We evaluated *bagging* and *different seeds* with three non-trivial applications of ILP.

Our results show that ensembles built through *bagging* can indeed achieve a sizable improvement in performance, both measured through accuracy and through ROC curves. Most of the gain is achieved with ensembles of size up to 20. Exploiting different seeds can also achieve very good gains. In fact, simply using *different seeds* worked as well, or arguably better, than *bagging* in our experiments. We believe this is because *different seeds* learns theories using the whole set of examples. Our results confirm the advantages of using ensemble methods in ILP.

We believe that *bagging* and *different seeds* can have a substantial impact on ILP. We can often achieve an interesting improvement in performance, with little implementation work. Moreover, we found that accuracy may improve, even in the cases where ILP is obtaining very good results, as is the case of the dataset Smuggling. On the other hand, resulting theories are more complex and thus harder to understand.

We have thus far used *bagging* and *different seeds* with ensembles of theories. An interesting alternative we are researching is to use ensembles of clauses. As discussed before, *boosting* can also be employed to improve accuracy of classifiers by penalizing examples that are misclassified. One disadvantage of *boosting* is that it can not be as easily parallelisable as *bagging* or *different seeds*. We have been investigating a method to perform *boosting* in parallel. Last, we are investigating other tuning algorithms to improve the interaction between different settings (e.g., clause length, minimum clause accuracy) for the ILP search and the *bagging*/*different seeds* process.

Acknowledgments

This work was supported by DARPA EELD grant number F30602-01-2-0571, the NSF grant 9987841 and by NLM grant NLM 1 R01 LM07050-01. Vítor Santos Costa and Inês de Castro Dutra were partially supported by CNPq. We would like to thank the Biomedical Group support staff for their invaluable help with Condor. We also would like to thank Ashwin Srinivasan for his help with the Aleph system and the Carcinogenesis benchmark. Vítor Santos Costa and Inês Dutra are on leave from COPPE/Sistemas, Federal University of Rio de Janeiro.

References

1. E. Alpaydin. Multiple networks for function learning. In *IEEE International Conference on Neural Networks*, pages 9–14, 1993.
2. J. Basney and M. Livny. Managing network resources in Condor. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania*, pages 298–299, Aug 2000.
3. H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Executing query packs in ILP. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 60–77. Springer-Verlag, 2000.
4. H. Blockeel, B. Demoen, G. Janssens, H. Vandecasteele, and W. Van Laer. Two advanced transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 43–59, 2000.
5. I. Bratko and M. Grobelnik. Inductive learning applied to program construction and verification. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 279–292. J. Stefan Institute, 1993.
6. L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
7. L. Breiman. Stacked Regressions. *Machine Learning*, 24(1):49–64, 1996.
8. L. Dehaspe and L. De Raedt. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.
9. T. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2000.
10. B. Dolšak and S. Muggleton. The application of ILP to finite element mesh design. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 225–242, 1991.
11. S. Džeroski, L. Dehaspe, B. Ruck, and W. Walley. Classification of river water quality data using machine learning. In *Proceedings of the 5th International Conference on the Development and Application of Computer Techniques to Environmental Studies*, 1995.
12. Y. Freund and R. Shapire. Experiments with a new boosting algorithm. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 148–156. Morgan Kaufman, 1996.
13. J. Graham, D. Page, and A. Wild. Parallel inductive logic programming. In *Proceedings of the Systems, Man, and Cybernetics Conference*, 2000.
14. L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990.
15. S. Hoche and S. Wrobel. Relational learning using constrained confidence-rated boosting. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 51–64. Springer-Verlag, September 2001.
16. R. King, S. Muggleton, and M. Sternberg. Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647–657, 1992.
17. A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.

18. N. Lincoln and J. Skrzypek. Synergy of clustering multiple backpropagation networks. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1989.
19. T. Matsui, N. Inuzuka, H. Seki, and H. Ito. Parallel induction algorithms for large samples. In S. Arikawa and H. Motoda, editors, *Proceedings of the First International Conference on Discovery Science*, volume 1532 of *Lecture Notes in Artificial Intelligence*, pages 397–398. Springer-Verlag, December 1998.
20. J. Metz. The epidemic in a closed population with all susceptibles equally vulnerable; some results for large susceptible populations and small initial infections. *Acta Biotheoretica*, 27:75–123, 1978.
21. D. W. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.
22. D. W. Opitz and J. W. Shavlik. Actively searching for an effective neural-network ensemble. *Connection Science*, 8(3/4):337–353, 1996.
23. F. J. Provost and T. Fawcett. Robust classification systems for imprecise environments. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 706–713, 1998.
24. J. R. Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the 14th National Conference on Artificial Intelligence*, volume 1, pages 725–730, 1996.
25. J. R. Quinlan. Boosting first-order learning. *Algorithmic Learning Theory, 7th International Workshop, Lecture Notes in Computer Science*, 1160:143–155, 1996.
26. V. Santos Costa, A. Srinivasan, and R. Camacho. A note on two simple transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 225–242. Springer-Verlag, 2000.
27. M. Sebag and C. Rouveirol. Tractable induction and classification in first-order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann, 1997.
28. A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.
29. A. Srinivasan. *The Aleph Manual*, 2001.
30. A. Srinivasan, R. King, S. Muggleton, and M. Sternberg. Carcinogenesis predictions using ILP. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 273–287. Springer-Verlag, 1997.
31. J. Struyf and H. Blockeel. Efficient cross-validation in ILP. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 228–239. Springer-Verlag, September 2001.
32. F. Zelezny, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *The Twelfth International Conference on Inductive Logic Programming*. Springer Verlag, July 2002.
33. J. Zelle and R. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 817–822, Washington, D.C., July 1993. AAAI Press/MIT Press.
34. S. Zemke. Bagging imperfect predictors. In *Proceedings of the International Conference on Artificial Neural Networks in Engineering, St. Louis, MI, USA*. ASME Press, 1999.
35. M. Zweig and G. Campbell. Receiver-operative characteristic. *Clinical Chemistry*, 39:561–577, 1993.

Lattice-Search Runtime Distributions May Be Heavy-Tailed

Filip Železný[1], Ashwin Srinivasan[2], David Page[3]

[1] Dept. of Cybernetics
Faculty of Electrical Engineering
Czech Technical University
Karlovo Nám. 13, 121 35 Prague, Czech Republic
zelezny@fel.cvut.cz
[2] Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
ashwin@comlab.ox.ac.uk
[3] Dept. of Biostatistics and Medical Informatics and Dept. of Computer Science
University of Wisconsin
1300 University Ave., Rm 5795 Medical Sciences
Madison, WI 53706, USA
page@biostat.wisc.edu

Abstract. Recent empirical studies show that runtime distributions of backtrack procedures for solving hard combinatorial problems often have intriguing properties. Unlike standard distributions (such as the normal), such distributions decay slower than exponentially and have "heavy tails". Procedures characterized by heavy-tailed runtime distributions exhibit large variability in efficiency, but a very straightforward method called *rapid randomized restarts* has been designed to essentially improve their average performance. We show on two experimental domains that heavy-tailed phenomena can be observed in ILP, namely in the search for a clause in the subsumption lattice. We also reformulate the technique of randomized rapid restarts to make it applicable in ILP and show that it can reduce the average search-time.

1 Introduction

In the recent paper [4], Gomes et al. observe that procedures for solving propositional satisfiability problems exhibit a remarkable runtime variability. The runtimes vary greatly depending on the choice of a particular heuristic, a given problem instance, or - for stochastic methods - on the choice of different random seeds (initial truth assignments), or on another source of randomness. Often a satisfiability procedure "hangs" on a given problem instance, while a different stochastic run solves the same instance quickly. Even for a deterministic procedure and a given problem

instance, small amount of randomization (e.g. in the employed heuristic) yields again largely varying search-costs, some of which are substantially lower than that of the deterministic algorithm.

In their empirical study it is shown that distributions of the runtimes of many search algorithms decay slower than exponentially and asymptotically have *heavy-tails*. Unlike standard probability distributions, such as the normal distribution, where events that are several standard deviations from the mean are very rare, in heavy-tailed distributions there is a non-negligible probability that an event with an extremely high cost occurs. For example, in one of the studied problems, 80% of the runs solve the problem in 1,000 backtracks or less, however 5% of the runs do not result in a solution even after 1,000,000 backtracks. Gomes et al. believe that the heavy-tailedness is a property of many exhaustive backtrack algorithms for solving hard combinatorial problems, and offer a technique called *randomized rapid restarts* that exploits this property in order to reduce the average search-time. The technique was used to find solutions of previously unsolved instances of hard combinatorial problems.

Although many search problems give rise to a heavy-tailed distribution, others do not [5]. Our aim is to find out whether heavy-tailed runtime distributions occur in ILP. Namely, we empirically study the runtimes of the search for a first-order clause with defined desired properties, in the lattice imposed by the subsumption relation. Furthermore we reformulate the randomized rapid restarts algorithm to be applicable on the ILP search problem and on two important ILP benchmarks we evaluate whether it reduces the search-cost with respect to a deterministic exhaustive search.

The following section defines formally the notion of a heavy-tailed distribution and describes the method of randomized rapid restarts. Since the method requires randomization of the exhaustive search, Section 3 describes our way of randomization of the lattice search, based on the selection of a random starting clause (seed). The core of the study is Section 4 which will test empirically the hypothesis that heavy-tailed runtime distributions describe the clause lattice-search using benchmark ILP problems. In the same section, we shall also apply the technique of randomized rapid restarts on the same domains, and investigate whether it improves search efficiency. We summarize our observations in Section 6.

2 Heavy-tailed Distributions and Randomized Rapid Restarts

The cumulative probability distribution $Pr(X < x)$ of a random variable X is a non-decreasing real function on the real interval $-\infty < x < \infty$ and will be denoted $F(x)$, i.e. $Pr(X > x) = 1 - F(x)$. Standard probability distributions have exponentially decreasing tails, e.g. for the standard normal distribution F_n it holds that

$$(1 - F_n(x)) \sim \frac{1}{x\sqrt{2\pi}} \exp \frac{-x^2}{2} \tag{1}$$

where $g(x) \sim h(x)$ denotes $\lim_{x\to\infty} g(x)/h(x) = 1$.

Recently, in the area of algorithms for hard combinatorial problems [4] but also other areas such as statistical physics, economics etc., different phenomena have been shown to obey heavy-tailed distributions which often lead to non-intuitive behaviour. For this class of distribution it holds that

$$(1 - F(x)) \sim Cx^{-\alpha}, x > 0 \tag{2}$$

where $0 < \alpha < 2$ and $C > 0$ are constants. It is remarkable [4] that such distributions have finite mean but no finite variance if $1 < \alpha < 2$. If $\alpha \leq 1$, the distribution has even neither a finite mean nor a finite variance.

To determine whether a distribution estimated by a series of measurement has a heavy-tailed nature, i.e. it does not decay exponentially, we plot the measured distribution values in a diagram with both axis logarithmically scaled, because an exponentially decreasing distribution should show a faster-than-linear decay in the log-log scale. For example, substituting x with $\exp(x)$ in the normal distribution decay (see Eq. 1) and taking log yields

$$\log\{1 - F_n(\exp[x])\} \sim -\left(x + \frac{\exp(2x)}{2}\right) \tag{3}$$

while the same operation on the heavy-tailed distribution decay (see Eq. 2) yields $-\alpha x$, i.e. a heavy-tailed distribution should exhibit an approximately linear decay on the log-log scale as x approaches infinity.

Let us now consider an exhaustive-search algorithm randomized in such a way, that it starts the search at a randomly chosen point of the search space (seed). Depending on the particular type of a search problem and algorithm, the distribution of times required to reach a solution from

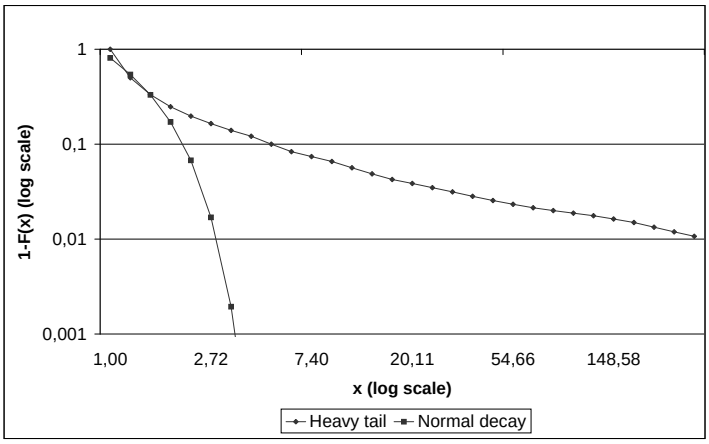


Fig. 1. Example of normal and heavy-tailed distributions on a log-log scale. The normal distribution decays faster than linearly while the heavy-tailed distribution decay shows an approximately linear decay.

such seeds may or may not be heavy-tailed.[1] If a heavy tail is observed, the runtime variance and mean may be infinite and there is a non-negligible probability that a chosen seed will start an extremely costly search, although many other seeds may produce a quick path to the solution. A direct way to reduce the variance and mean in such a case is to run the exhaustive search up to a certain cutoff point and then restart at a different seed if a solution is not found. Clearly, this approach called *randomized rapid restarts* avoids the algorithm from getting trapped in a very costly path and exploits the high chance of obtaining an essentially shorter path in the next trial. It is shown in [4] that the randomized rapid restarts technique is superior to deterministic exhaustive searches in many propositional domains.

[1] Gomes et al [5] discover the heavy-tailedness of different search problems purely empirically and report that further studies are needed to determine exactly what characteristics of combinatorial search problem lead to heavy-tailed behaviour. In this ILP-focused study we also take an empirical approach.

3 Randomizing the Lattice Search

We consider the normal ILP problem [10], namely we assume the sets of positive (negative) examples E^+ (E^-) and background knowledge B. We also assume that a search lattice of legal clauses has been defined by the generality (subsumption) relation, a clause mode language $\mathcal{L}$, and bounded by the most specific element $\perp$ (the bottom clause). This is a usual assumption of ILP systems based on the concept of inverse entailment [9], such as Aleph [6] and Progol [9].

The standard approach of conducting the lattice search is to start with the most general (most specific, $\perp$) clause and then proceed in a top-down (bottom-up) manner. However, starting the search with a different clause from the interior of the lattice may lead to a shorter runtime needed to reach the desired clause. Our plan is to investigate the distribution of such runtimes when the starting clauses are selected randomly from the lattice. If this distribution proves to be heavy-tailed, we will be able to utilize the technique of randomized rapid restarts to avoid the extreme-cost paths and improve the average performance.

The randomized search algorithm proceeds as follows. Some number of times (maxtries), the algorithm will carry out a short search (bounded by maxtime, Section 4). Each search begins by stochastic selection of a starting clause. The search is a deterministic best-first search, with heuristic function $h = pos(C) - neg(C)$. Here $pos(C)$ is the number of positive examples deducible from $C \wedge B$[2], and $neg(C)$ is analogous for negative examples. From a given clause C, the neighbors of C in the search space are defined by a nontraditional refinement operator ρ. It differs from usual refinement operators employed in the top-down search of the mentioned systems in that it produces the set $REFS = \rho(C)$ of all neighbours of C in the lattice, i.e. also including clauses that subsume C. Therefore this kind of search can be seen as radial, rather than top-down or bottom-up (visualized in Fig. 2). With this refinement operator, all nodes in the lattice can be reached from any starting node.

We shall now address the problem of choosing the seed, i.e. the initial stochastic clause selection. The principal difficulty of its implementation lies in devising a procedure for uniform random sampling of clauses from the search space. Here, we describe a procedure (from [14]) that does not require prior generation of all elements of the search space. Recall that these are definite clauses obtained from subsets of literals drawn from a

[2] In the case when C is constructed as an extension of an existing partial theory H in a greedy cover search, we assume that H has been added to B.

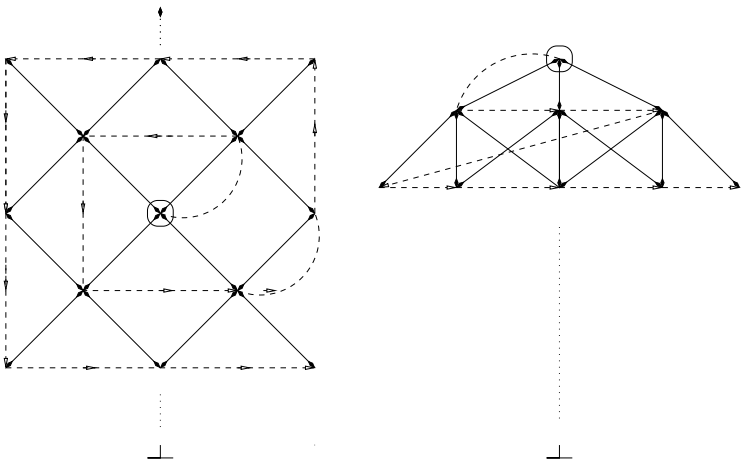


Fig. 2. A schematic visualization of the radial lattice search (left) compared to a top-down search (right). Nodes are explored starting at the encircled point and then following the dashed line. This view is simplified in that the employed heuristic function $h(C)$ is assumed to be constant for all C and descendants of explored nodes are inserted in the end of the *open* list, which in the top-down case corresponds to a breadth-first search.

most specific (definite) clause $\perp$. Additional provisos are that each subset is of cardinality at most $c+1$ (where c is a user-specified maximum number of negative literals) and is in the language $\mathcal{L}$. Let $\mathcal{C}$ denote all such clauses. Further, let the number of clauses in $\mathcal{C}$ with exactly l literals be n_l and $\mathcal{N}$ denote the subset of natural numbers $\{1,\ldots,|\mathcal{C}|\}$. Define a function $h : \mathcal{C} \rightarrow \mathcal{N}$ such that $h(C) = \sum_{i=1}^{|C|-1} n_i + j$ where $|C|$ is the number of literals in C and $1 \leq j \leq n_{|C|}$. That is, h provides a sequential enumeration of clauses by length. While many functions fit this requirement (depending on the enumeration adopted), it is easy to show that any such h is both $1-1$ and *onto*. It follows that h is invertible – that is, given a number in $\mathcal{N}$, it is possible to find a unique clause in $\mathcal{C}$ provided the n_i (and c) are known. In principle, it is therefore possible to achieve the selection required by randomly choosing a number n in $\mathcal{N}$ and returning $C = h^{-1}(n)$. Such an inverse function works as follows. Given a number $n > 0$: (a) find the largest number $l = 0 \ldots c$ such that $j = n - \sum_{i=0}^{l} n_i > 0$; (b) generate a sequence of clauses in $\mathcal{L}$ of length $l+1$. C is the j^{th} clause in this sequence. If n is randomly generated, then the clause generation process does not have to be so, and can be made more efficient by various devices. Some examples are: (a) take C to be

estimate$(\bot, \mathcal{L}, l, s)$: Given a most specific clause $\bot$ and a clause length $l > 1$, returns an estimate of the number of definite clauses of length l in $\mathcal{L}$ such that each clause is a subset of $\bot$. The estimate is obtained from a sample of size s.

1. Sample s clauses of length l from $\bot$. Each such clause consists of the positive ("head") literal in $\bot$ and a random selection, without replacement, of $l-1$ literals from the negative ("body") literals in $\bot$.
2. Determine the proportion p_l of the s clauses that are in $\mathcal{L}$.
3. return $p_l \times (|\bot| - 1) \times \ldots \times (|\bot| - l + 1)$

Fig. 3. A procedure for estimating the number of "legal" clauses of length $l > 1$. The estimate obtained in Step 3 above is unbiased [18]. The value of the sample size s needs to be decided. An option is to be guided by statistical estimation theory. This states that if values of p_l are not too close to 0 or 1, then we can be at least $100 \times (1-\alpha)\%$ confident that the error will be less than a specified amount e when $s = z^2_{\alpha/2}/(4e^2)$ [18]. Here z represents the standard normal variable as usual.

the first clause of that length (and in $\mathcal{L}$) that has not been drawn before; (b) a once-off generation of the appropriate number of clauses in $\mathcal{L}$ at each length ("appropriate" here means that the proportion of clauses of length i in the sample is $n_i/|\mathcal{N}|$); and (c) using a dependency graph over literals in $\bot$ to ensure that the random clause construction always results in clauses within the language $\mathcal{L}$.

In practice, without prior generation of the set $\mathcal{C}$, the n_i are not known for $i > 1$ and we adopt the procedure in Fig. 3[3] for estimating them.

Having constructed a method for stochastic selection of the starting clause and its subsequent deterministic refinement, we are in a position to implement the technique of randomized rapid restarts. An implementation for the clause lattice search is described in Fig. 4. The underlying principle that makes the technique applicable to the clause search is that, unlike in usual ILP approaches, we do not search through a specified number of nodes, returning the best clause found, but rather we stop the search once a clause is found meeting a pre-specified condition of 'goodness' as follows.

$$pos(C) > 1 \tag{4}$$

$$\frac{pos(C)}{pos(C) + neg(C)} > Acc \tag{5}$$

where $pos(C)$ ($neg(C)$) is the number of positives (negative) examples

[3] This method is implemented in the ILP system Aleph [6] and can serve as well for other methods of randomized local search, such as GSAT or WSAT

$rrr(Lat, Acc, B, E^+, E^-, maxtime, maxtries)$: Given background knowledge B, positive and negative examples E^+, E^-, return a clause from the given subsumption lattice Lat, that satisfies the conditions in Eqs. 4 and 5 for the given constant Acc.

1. $tries := 1$
2. Select a random starting clause C_0 using the procedure described in Section 3.
3. $searchtime := 0$, start timing.
4. Starting at C_0, perform an exhaustive radial search described in Section 3, until $searchtime > maxtime$ or a clause C satisfying Eqs. 4, 5 is found.
5. If C was found, STOP, return C.
6. If $tries < maxtries$, $tries := tries + 1$, Go to 2. Otherwise return "failure".

Fig. 4. An implementation of randomized rapid restarts for the clause lattice search. The maximum time for one exhaustive search is limited by the constant $maxtime$. The maximum number of repeated searches is $maxtries$.

covered by C. The first condition avoids the trivial solution $C = e, e \in E^+$ and the second condition is parameterized by a constant Acc.

We set the maximum number of allowed restarts $maxtries$, which should theoretically be infinite, to a finite number ($maxtries = 10$) because we cannot guarantee exclude that a clause satisfying the prespecified condition exists in the lattice. In the case of reaching the limit $maxtries$ ("failure" in Fig. 4), the positive example used to construct the current bottom clause $\perp$ is returned as the resulting clause. The setting of $maxtime$ will be discussed in connection with the experimental setting in Section 4.

Finally, to cover all of the positive examples in the training sets of the experiments, we use the greedy covering approach as usual in ILP systems, i.e. the procedure in Fig. 4 is run repeatedly with a bottom clause constructed using one selected positive, until all positives are covered, each time adding the newly constructed clause to the background knowledge and deleting the covered positives from the training set.

4 Experiments

4.1 The Aim

We shall investigate (a) if the heavy-tailed phenomenon manifests itself when searching the subsumption lattice; and (b) if utilizing RRR will help improve search efficiency for problems exhibiting the heavy-tailed phenomenon.

4.2 Materials

Our experimental material consists of two sets of pre-classified data, namely the data on the mutagenic and carcinogenic properties of some chemicals. These data are publicly available (anonymous ftp to `ftp.comlab.ox.ac.uk` in the directories `pub/Packages/ILP/Datasets/mutagenesis/aleph` and `pub/Packages/ILP/Datasets/carcinogenesis/aleph`).
We refer the reader to [16, 17] for detailed descriptions of background knowledge available for the mutagenesis task. The background information is encoded in approximately 13,000 facts. The background knowledge for the carcinogenesis problem is conceptually of a similar nature. The encoding requires approximately 24,500 facts – see [15] for more details.

All of our subsequent experiments use an Athlon 1500MHz CPU – based computer with 512KB of RAM and the ILP program Aleph (Version 3). Aleph is available at: `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.pl`.

The language $\mathcal{L}$ will be limited to clauses of maximum number of 4 negative literals and maximum variable depth [9] 2.

4.3 Methods and Results

We shall observe the stochastic behaviour of the randomized search-algorithm described in the previous section, namely the distribution of search-times required to find a clause C which satisfies the conditions in Eqs. 4,5, where we set $Acc = 0.7$.

Figure 5 shows the cumulative distribution $F(x)$ of runtimes for the mutagenesis task, Figure 6 an analogous distribution for the carcinogenesis task. Both of the distributions are collected from about 35,000 searches starting in random seeds. In the mutagenesis task, for example, almost 20% of runs arrive at a solution in less then 0.1s, however almost 30% of runs do not find a solution in 20s.

Figures 7 and 8 clearly show that both of the experimentally measured distributions exhibit a heavy-tail (c.f. Section 2). According to the study [4], our findings justify the application of the method of randomized rapid restarts. To use the algorithm described in Section 3, we need to set the cutoff value $maxtime$. There is no analytic way of determining the optimal value $maxtime_{opt}$ of the cutoff value, but it is reported [4] to lie below the median point of the runtime distribution $F(x)$, i.e. $maxtime_{opt} << 1s$ for both our experimental domains. As it may be infeasible in the general case to construct the runtime distributions $F(x)$ for a given problem prior to the learning process, we shall disregard the information provided by

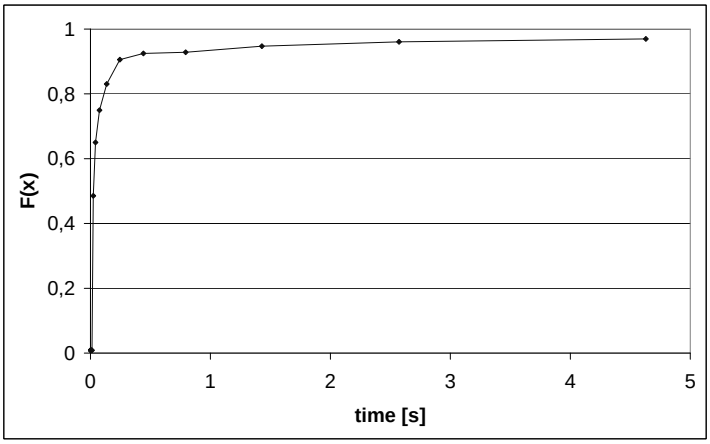


Fig. 5. The cumulative distribution $F(x)$ of runtimes of the randomized algorithm searching for a 'good' clause in the mutagenesis problem.

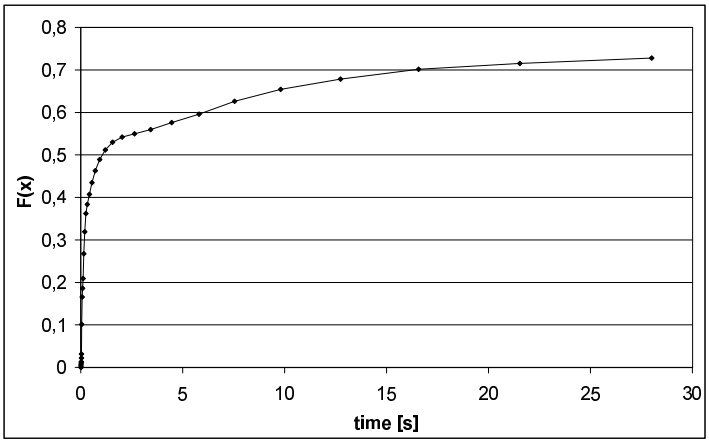


Fig. 6. The cumulative distribution $F(x)$ of runtimes of the randomized algorithm searching for a 'good' clause in the carcinogenesis problem.

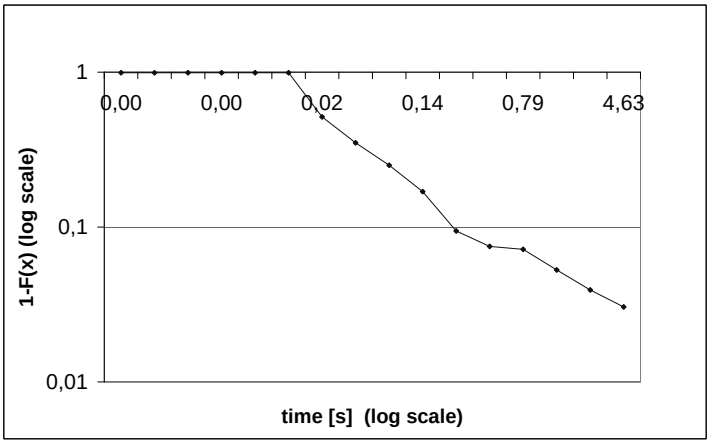


Fig. 7. A log-log plot of the distribution decay $1 - F(x)$ concerning the same data as in Figure 5.

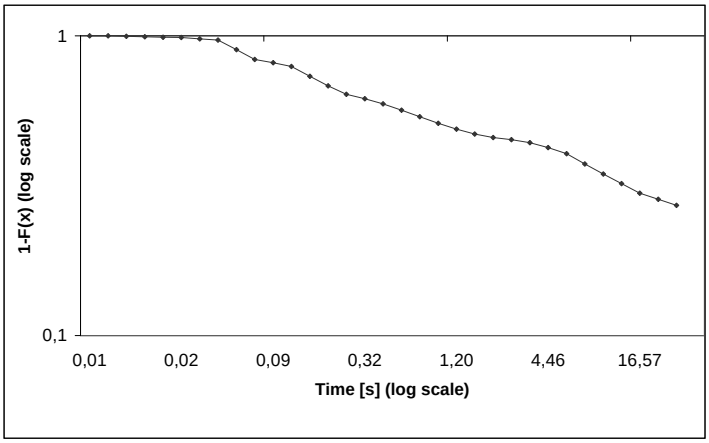


Fig. 8. A log-log plot of the distribution decay $1 - F(x)$ concerning the same data as in Figure 6.

the already generated distributions, and we choose a small *ad hoc* value $maxtime = 1s$ for both of the domains in the comparative experiments.

Table 1 summarizes the predictive accuracies and learning times of the randomized rapid restarts technique vs. the standard exhaustive breadth-first top-down search algorithm. The former method was tested with the *Acc* parameter set to the values 0.7 and 0.9. Similarly, the latter method was tested with two values 0.7 and 0.9 of the *minimum accuracy* requirement on a clause to be accepted for the constructed theory.

The results suggest that by using randomized rapid restarts we achieved a drastic reduction of the search times for the price of only a small loss in predictive accuracy.

Algorithm	MUT		CANC	
	A (%)	T (s)	A (%)	T (s)
DTD 0.7	88.76	1589	57.91	24092
	(5.99)	(461)	(9.75)	(11915)
DTD 0.9	88.23	1541	56.22	22101
	(5.63)	(459)	(8.98)	(9811)
RRR 0.7	87.71	9	54.84	74
	(7.62)	(4)	(8.97)	(10)
RRR 0.9	86.31	24	57.57	126
	(8.67)	(10)	(6.39)	(71)

Table 1. Estimated predictive accuracies (A) and theory construction times (T). The entries are from a 10-fold cross-validation design with time entries rounded up to the nearest second. The numbers in parentheses are estimates of standard deviation. These are obtained by a simple binomial formula that ignores the dependencies across cross-validation runs. Exact calculation of standard deviations for results from cross-validation designs is confounded by these dependencies but the approximation used here has been found to be adequate (see [1], pg 307). The algorithms result from two search techniques employed by Aleph, namely: deterministic top-down (DTD) (with minimum clause accuracy setting 0.7 and 0.9, respectively) and randomized rapid restarts (with clause threshold 0.7 and 0.9, respectively). MUT refers to the mutagenesis problem, CANC to carcinogenesis. The search space is limited by the maximum clause-length of 5 literals and maximum variable depth 2.

5 Related Work

Besides the direct inspiration by the findings due to Gomes et al., this work is also related to the research of the *phase transition* effect. Phase transition has been observed in algorithms for solving difficult computa-

tional problems, namely NP-complete ones such as the constraint satisfaction problem (CSP) [13]. A *constraint tightness* parameter $p \in < 0; 1 >$ can be calculated for any CSP instance. According to empirical studies [13], the expected time to solve a CSP is small for values of p close to 0 (phase of 'many solutions available') or 1 (phase of 'inspecting a small search tree') and grows dramatically for p close to a *critical value* p_{cr} (transition between the two phases). In the surrounding of p_{cr}, the costs of solving CSP instances not only show a high mean, but also a large variability. Frost el al. [2] approximate the cost distributions of instances with p close to p_{cr} with various closed-form distributions. They point out (independently of Gomes el al.) the long tails of these distributions and report that "problems that are not solved early are likely to take a long time". The fundamental bridge between such findings and ILP is the fact that the first-order subsumption problem can be mapped onto a CSP [7]. Botta et al. then show [8] that a typical ILP program (FOIL) tends to "induce hypotheses generating matching problems located inside the phase transition region"; Giordana and Saitta report a similar observation [3] in real-world domains, including mutagenesis. Combining the referred results, the heavy-tailed effect had been expectable before our study, which can thus be seen as an empirical verification of this implicit expectation. Unlike the previous studies where statistics were measured for a collection of the proving (subsumption check) problem instances, we measured distributions on a collection of complete hypothesis-searching cycles, each containing a number of subsumption tests.

The way statistical observations are exploited towards efficiency improvements also distinguishes our approach from the mentioned related work. Sebag and Rouveirol [11] apply a stochastic algorithm in order to accelerate the subsumption-test and Giordana and Saitta [3] develop an on-line complexity estimator which can potentially be used for the same purpose. Our approach, on the other hand, allows to adopt the RRR technique to reduce the complexity of the hypothesis search in its entirety.

As far as the randomized technique of traversing through the subsumption lattice is concerned, to our best knowledge, there is only indirectly related work to our study. Serra et al [12] show that starting the search for a hypothesis from random seed formulas, instead than top-down, can be beneficial. Randomized search in an ILP system has been assessed in [14].[4]

[4] We are also aware of the talk of Stephen Muggleton at the Machine Intelligence workshop in 2001 about randomization techniques in Progol but as we gather, there is no written account on that talk.

6 Conclusions

Our study has shown that the phenomenon of heavy-tailed runtime distributions occurs in two important experimental domains of ILP and we believe that it is typical to many other domains. Testing this hypothesis is a part of our future work.

This observation lead to the utilization of the technique of randomized rapid restarts which we reformulated for sakes of ILP. To apply this method, the exhaustive lattice search was randomized in such a way that we selected randomly the clause where the search was started. Randomized rapid restarts may then be used to reduce the average time required to find a clause with desired properties. A natural question is whether reducing the average runtime of the search procedure randomized in this way may lead to outperforming the deterministic top-down or bottom-up search. But clearly, if we do not impose a prior probability distribution on clauses (or e.g. on clause-lengths), there is no reason to expect that a search starting from the most general (most specific) element will be systematically faster than the average search starting in a random element of the lattice.

Using the technique of randomized rapid restarts, we were able to significantly reduce the search times in large hypothesis spaces of both of the tested domains.

7 Acknowledgements

We thank the ILP'02 referees for pointing us to some very relevant articles. Also, the ILP'02 audience contributed much to the paper by motivating us to relate our study to the phase transition research. Filip Zelezny greatly acknowledges the support from the EU project INCO 977102 ILPnet2 and the Czech Technical University grant CTU 0209013. David Page was supported by the U.S. National Science Foundation grant 9987841 and a U.S. DARPA EELD grant.

References

1. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
2. Daniel Frost, Irina Rish, and Lluis Vila. Summarizing CSP hardness with continuous probability distributions. In *AAAI/IAAI*, pages 327–333, 1997.
3. A. Giordana and L. Saitta. Phase transitions in relational learning. *Machine Learning*, 2000.

4. C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.
5. C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.
6. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html.
7. J. Maloberti and M. Sebag. Theta-subsumption in a constraint satisfaction perspective. volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 164–178. Springer-Verlag, September 2001.
8. M.Botta, A.Giordana, L.Saitta, and M.Sebag. Relational learning: hard problems and phase transitions. In *6th Congress of the Italian Association for Artificial Intelligence*. Springer-Verlag, 1999.
9. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
10. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
11. Michele Sebag and Celine Rouveirol. Resource-bounded relational reasoning: Induction and deduction through stochastic matching. *Machine Learning*, 38(1/2):41–62, January 2000.
12. A. Serra, A. Giordana, and L. Saitta. Learning on the phase transition edge. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 921–926. Morgan Kaufmann, 2001.
13. Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.
14. A. Srinivasan. A study of two probabilistic methods for searching large spaces with ILP. Technical Report PRG-TR-16-00, Oxford University Computing Laboratory, Oxford, 2000.
15. A. Srinivasan and R. King. Carcinogenesis predictions using ilp. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 3–16. Springer-Verlag, 1997.
16. A. Srinivasan and R.D. King. Feature construction with Inductive Logic Programming: a study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.
17. A. Srinivasan, S. Muggleton, M.J.E. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1,2), 1996.
18. R. Walpole and R. Myers. *Probability and Statistics for Engineers and Scientists*. Collier Macmillan, New York, 1978.

In Data Mining: Next Generation Challenges and Future Directions,
H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (Eds.), AAAI Press, 2004

Chapter 1

Relational Data Mining with Inductive Logic Programming for Link Discovery

Raymond J. Mooney[*], *Prem Melville*[*], *Lappoon Rupert Tang*[*], *Jude Shavlik*[‡], *Inês de Castro Dutra*[‡], *David Page*[‡], *Vítor Santos Costa*[‡]

[*]Department of Computer Sciences
University of Texas
Austin, TX 78712-1188
{mooney,melville,rupert}@cs.utexas.edu

[‡]Department of Biostatistics and Medical Informatics and
Department of Computer Sciences
University of Wisconsin
Madison, WI 53706-1685
{shavlik,dpage}@cs.wisc.edu, {dutra,vitor}@biostat.wisc.edu

Abstract:
Link discovery (LD) is an important task in data mining for counter-terrorism and is the focus of DARPA's Evidence Extraction and Link Discovery (EELD) research program. Link discovery concerns the identification of complex relational patterns that indicate potentially threatening activities in large amounts of relational data. Most data-mining methods assume data is in the form of a feature-vector (a single relational table) and cannot handle multi-relational data. *Inductive logic programming* is a form

f relational data mining that discovers rules in first-order logic from multi-relational data. This paper discusses the application of ILP to learning patterns for link discovery.

Keywords: Relational Data Mining, Inductive Logic Programming, counter-terrorism, link discovery

1.1 Introduction

Since the events of September 11, 2001, the development of information technology that could aid intelligence agencies in their efforts to detect and prevent terrorism has become an important focus of attention. The Evidence Extraction and Link Discovery (EELD) program of the Defense Advanced Research Projects Agency (DARPA) is one research project that attempts to address this issue. The establishment of the EELD program for developing advanced software for aiding the detection of terrorist activity pre-dates the events of 9/11. The program had its genesis at a preliminary DARPA planning meeting held at Carnegie Mellon University after the opening of the Center for Automated Learning and Discovery in June of 1998. This meeting discussed the possible formation of a new DARPA research program focused on novel knowledge-discovery and data-mining (KDD) methods appropriate for counter-terrorism.

The scope of the new program was subsequently expanded to focus on three related sub-tasks in detecting potential terrorist activity from numerous large information sources in multiple formats. *Evidence Extraction* (EE) is the task of obtaining structured evidence data from unstructured, natural-language documents. EE builds on information extraction technology developed under DARPA's earlier MUC (Message Understanding Conference) programs [Lehnert & Sundheim1991, Cowie & Lehnert1996] and the current ACE (Automated Content Extraction) program at the National Institute of Standards and Technology (NIST)[NIST2003]. *Link Discovery* (LD) is the task of identifying known, complex, multi-relational patterns that indicate potentially threatening activities in large amounts of relational data. It is therefore a form of pattern-matching that involves matching complex, multi-relational "patterns of interest" against large amounts of data. Some of the input data for LD comes from EE applied to news reports and other unstructured documents, other input data comes from existing relational databases on financial and other transactions. Finally, *Pattern Learning* (PL) concerns the automated discovery of new relational patterns for potentially threatening activities. Since determining and authoring a complete and accurate set of formal patterns for detecting terrorist activities is a difficult task, learning methods may be useful for automatically acquiring such patterns from supervised or unsupervised data. Learned patterns can then be employed by an LD system to improve its detection of threatening activities. The current EELD program focused on these three sub-topics started in the summer of 2001. After 9/11, it was incorporated under the new Information Awareness Office (IAO) at DARPA.

The data and patterns used in EELD include representations of people, organizations, objects, and actions and many types of relations between them. The data is perhaps best represented as a large graph of entities connected by a variety of relations.

The areas of *link analysis* and *social network analysis* in sociology, criminology, and intelligence [Jensen & Goldberg1998, Wasserman & Faust1994, Sparrow1991] study such networks using graph-theoretic representations. Data mining and pattern learning for counter terrorism therefore requires handling such multi-relational, graph-theoretic data.

Unfortunately, most current data-mining methods assume the data is from a single relational table and consists of flat tuples of items, as in market-basket analysis. This type of data is easily handled by machine learning techniques that assume a "propositional" (a.k.a "feature vector" or "attribute value") representation of examples [Witten & Frank1999]. *Relational data mining* (RDM) [Džeroski & Lavrač2001b], on the other hand, concerns mining data from multiple relational tables that are richly connected. Given the style of data needed for link discovery, pattern learning for link discovery requires *relational* data mining. The most widely studied methods for inducing relational patterns are those in *inductive logic programming* (ILP) [Muggleton1992, Lavrac & Dzeroski1994]. ILP concerns the induction of Horn-clause rules in first-order logic (i.e., logic programs) from data in first-order logic. This paper discusses our on-going work on applying ILP to pattern learning for link discovery as a part of the EELD project.

1.2 Inductive Logic Programming (ILP)

ILP is the study of learning methods for data and rules that are represented in first-order predicate logic. Predicate logic allows for quantified variables and relations and can represent concepts that are not expressible using examples described as feature vectors. A relational database can be easily translated into first-order logic and be used as a source of data for ILP [Wrobel2001]. As an example, consider the following rules, written in Prolog syntax (where the conclusion appears first), that define the uncle relation:

```
uncle(X,Y) :- brother(X,Z),parent(Z,Y).
uncle(X,Y) :- husband(X,Z),sister(Z,W),parent(W,Y).
```

The goal of *inductive logic programming* (ILP) is to infer rules of this sort given a database of background facts and logical definitions of other relations [Muggleton1992, Lavrac & Dzeroski1994]. For example, an ILP system can learn the above rules for uncle (the *target predicate*) given a set of positive and negative examples of uncle relationships and a set of facts for the relations parent, brother, sister, and husband (the *background predicates*) for the members of a given extended family, such as:

```
uncle(tom,frank), uncle(bob,john),
not uncle(tom,cindy), not uncle(bob,tom)
parent(bob,frank), parent(cindy,frank), parent(alice,john),
parent(tom,john), brother(tom,cindy), sister(cindy,tom),
husband(tom,alice), husband(bob,cindy).
```

Alternatively, rules that logically define the brother and sister relations could be supplied and these relationships inferred from a more complete set of facts about only the "basic" predicates: `parent`, `spouse`, and `gender`.

If-then rules in first-order logic are formally referred to as *Horn clauses*. A more formal definition of the ILP problem follows:

- **Given:**
  - Background knowledge, B, a set of Horn clauses.
  - Positive examples, P, a set of Horn clauses (typically ground literals).
  - Negative examples, N, a set of Horn clauses (typically ground literals).
- **Find:** A hypothesis, H, a set of Horn clauses such that:
  - $\forall p \in P : H \cup B \models p$ (completeness)
  - $\forall n \in N : H \cup B \not\models n$ (consistency)

A variety of algorithms for the ILP problem have been developed [Džeroski & Lavrač2001a] and applied to a variety of important data-mining problems [Džeroski2001, Houstis *et al.*2000]. Nevertheless, relational data mining remains an under-appreciated topic in the larger KDD community. For example, recent textbooks on data mining [Han & Kamber2001, Witten & Frank1999, Hand, Mannila, & Smyth2001] hardly mention the topic. An increasing number of applications require handling complex, structured data types, including bioinformatics, web and text mining, and engineering. Therefore, we believe it is an important topic for "next generation" data mining systems. In particular, it is critical for link discovery applications in counter-terrorism.

One of the standard criticisms of ILP methods from a data-mining perspective is that they do not scale to large amounts of data. Since the hypothesis space of possible logic programs is extremely large and since just testing individual hypotheses requires potentially complex automated deduction, ILP methods can have difficulty processing large amounts of data. We have developed methods to help address both of these aspects of computational complexity. First, we have developed methods for controlling the number of hypotheses tested by developing new search methods that use stochastic search to more efficiently explore the space of hypotheses [Zelezny, Srinivasan, & Page2002] or that combine aspects of existing top-down and bottom-up methods (see section 1.3.2). Second, we have developed methods for automatically optimizing learned clauses by inserting cuts[1] in the Prolog code so that deduction is more efficient [Santos Costa, Srinivasan, & Camacho2000]. However, as discussed in section 1.4, scaling ILP to very large data sets is a significant area for future research.

1.3 Initial Work on ILP for Link Discovery

We tested several ILP algorithms on various EELD datasets. The current EELD datasets pertain to two domains that were chosen as "challenge problems" in link discovery that

[1] In Prolog, cuts (!) are procedural operators that prevent potentially computationally expensive backtracking where the programmer determines it is unnecessary.

have many of the underlying properties of the counter-terrorism problem – Nuclear Smuggling and Contract Killing. The Contract-Killing domain is further divided into natural (real world) data manually collected and extracted from news sources and synthetic (artificial) data generated by simulators. Section 1.3.1 presents our experimental results on the natural Smuggling and Contract-Killing data, while section 1.3.2 presents results on the synthetic Contract-Killing data.

1.3.1 Experiments on Natural Data

The Nuclear-Smuggling Data

The Nuclear-Smuggling dataset consists of reports on Russian nuclear materials smuggling [McKay, Woessner, & Roule2001]. The Chronology of Nuclear and Radioactive Smuggling Incidents is the basis for the analysis of patterns in the smuggling of Russian nuclear materials. The information in the Chronology is based on open-source reporting, primarily World News Connection (WNC) and Lexis-Nexis. There are also some articles obtained from various sources that have been translated from Italian, German and Russian. The research from which the Chronology grew began in 1994 and the chronology itself first appeared as an appendix to a paper by Williams and Woessner in 1995 [Williams & Woessner1995b, Williams & Woessner1995a]. The continually evolving Chronology then was published twice as separate papers in the same journal as part of the "Recent Events" section [Woessner1995, Woessner1997]. As part of the EELD project, the coverage of the Chronology was extended to March 2000 and grew to include 572 incidents.

The data is provided in the form of a relational database. This database contains Objects (described in rows in tables), each of which has Attributes of differing types (i.e., columns in the tables), the values of which are provided by the source information or the analyst. The Objects are of different types, which are denoted by prefixes (E_, EV_, and LK_), and consist of the following:

- Entity Objects (E_...): these consist of E_LOCATION, E_MATERIAL, E_ORGANIZATION, E_PERSON, E_SOURCE, and E_WEAPON;
- Event Objects (EV_...): these currently consist of the generic EV_EVENT;
- Link Objects (LK_...): used for expressing links between/among Entities and Events,

The database has over 40 relational tables. The number of tuples in a relational table varies from as many as 800 to as little as 2 or 3.

As a representative problem, we used ILP to learn rules for determining which events in an incident are *linked*. Such rules could be used to construct larger knowledge structures that could be recognized as threats. Hence, the ILP system is given positive training examples of known "links" between events. We assume all other events are unrelated and therefore compose a set of negative examples. We also provide background knowledge that the *linked* relation is commutative. Our training set consists of 140 positive examples and 140 distinct negative examples randomly drawn from a full set of 8,124 negative pairs of events. The linking problem in the Nuclear-Smuggling

data is thus quite challenging in that it is a heavily relational learning problem over a large number of relations, whereas traditional ILP applications usually require a small number of relations.

The Natural Contract-Killing Data

The dataset of contract killings was first compiled by O'Hayon and Cook [Cook & O'Hayon 2000] in response to research on Russian organized crime that encountered frequent references to contract killings. The dataset was subsequently expanded by the authors with funding from the EELD program through Veridian Systems Division [Williams2002]. The database consists of a chronology of incidents each described using information drawn from one or more news articles. As in the Nuclear-Smuggling dataset, information in the chronology is based on open-source reporting, especially Foreign Broadcast Information Service (FBIS) and Joint Publications Research Service (JPRS) journals, and subsequently both FBIS on-line and the on-line version World News Connection (WNC). These services and Lexis-Nexis were the main information sources.

The data is organized in relational tables in the same format as the Nuclear-Smuggling data described in the previous section. The dataset used in our experiments has 48 relational tables. The number of tuples in a relational table varies from as many as 1,000 to as few as 1. Each killing was categorized according to one of three possible motivations: "Rival," "Obstacle," or "Threat." The ILP task was to determine whether the motivation for a killing was categorized as "Rival" or not. The motivation for this learning task was to recognize patterns of activity that indicate an underlying motive, which in turn contributes to recognizing threats. The number of positive examples in this dataset is 38, while the number of negative examples is 34.

ILP Results on the Natural Data

ALEPH We used the ILP system ALEPH [Srinivasan2001] to learn rules for the natural datasets. By default, ALEPH uses a simple greedy set covering procedure that constructs a complete and consistent hypothesis one clause at a time. In the search for any single clause, ALEPH selects the first uncovered positive example as the seed example, *saturates* this example, and performs an admissible search over the space of clauses that subsume this saturation, subject to a user-specified clause length bound. Further details about our use of ALEPH in these experiments are available from [de Castro Dutra *et al.*2002].

Ensembles *Ensembles* aim at improving accuracy through combining the predictions of multiple classifiers in order to obtain a single classifier. In contrast with previous approaches [Quinlan1996, Hoche & Wrobel2001], each classifier is a logical theory generated by ALEPH. Many methods have been presented for ensemble generation [Dietterich1998]. We use *bagging* [Breiman1996a], a popular method that is known to frequently create a more accurate ensemble. Bagging works by training each classifier on a random sample from the training set. Bagging has the important advantage that it is effective on "unstable learning algorithms" [Breiman1996b], where small variations in the input data can cause large variations in the learned theories. Most ILP algorithms are unstable in this sense. A second advantage is that the bagging algorithm is highly

```
linked(A,E) :-
   lk_event_person(_,EventA,PersonC,_,RelationB,RelationB,DescriptionD),
   lk_event_person(_,EventF,PersonC,_,RelationB,RelationB,DescriptionD),
   lk_material_location(_,MaterialG,_,EventE,_,_,_,_,_),
   lk_event_material(_,EventF,MaterialG,_,_,_,_).
```

Figure 1.1: Nuclear-Smuggling Data: Sample Learned Rule

parallel [Dutra *et al.*2003]. Further details about our approach to bagging for ILP, as well as our experimental methodology, can be found in [de Castro Dutra *et al.*2002]. Our experimental results are based on a five-fold cross-validation, where five times we train on 80% of the examples and then test what was learned on the remaining 20% (in addition, each example is in one and only one test set).

For the task of identifying linked events in the Nuclear-Smuggling dataset, ALEPH produces an average testset accuracy of 85%. This is an improvement over the baseline case (majority class—always guessing two events are not linked), which produces an average accuracy of 78%. Bagging (with 25 different sets of rules) increases the accuracy to almost 90%.

An example of a rule with good accuracy found by the system is shown in Figure 1.1. This rule covers 43 of the 140 positive examples and no negative examples. According to this rule, two smuggling events A and E are related if event A involves a person C who is also involved in another event F. Event F involves some material G that appears in event E. In other words, a person C in event A is involved in a third event F that uses material from event E. Person C played the same role B, with description D, in events A and F. The "_" symbols mean that those arguments were not relevant for that rule. Figure 1.2 illustrates the connections between events, material and people involved. Solid lines are direct connections shown by the literals in the body of the clause. The dotted line corresponds to the newly learned concept that describes a connection between two events.

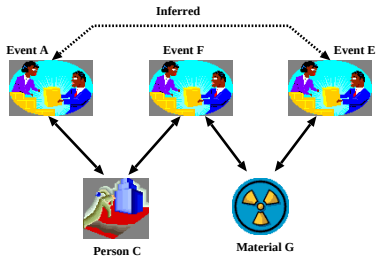


Figure 1.2: Pictorial representation of a learned rule.

The task of identifying the underlying motive in the Contract-Killing data set is much more difficult, with ALEPH's accuracy at 59%, compared with the baseline accuracy of 52%. Again, bagging improves the accuracy, this time to 69%. The rule in Figure 1.3 shows one logical clause the ILP system found for this dataset. The rule covers 19 of the 38 positive examples and a single negative example. The rule says that

```
rivalKilling(EventA) :-
     lk_event_event(_,EventB,EventA,RelationC,EventDescriptionD),
     lk_event_event(_,EventB,EventE,RelationC,EventDescriptionD),
     lk_event_event(_,EventE,EventF,_,EventDescriptionD),
     lk_org_org(_,_,_,EventF,_,_,_,_,_).
```

Figure 1.3: Natural Contract-Killing Data: Sample Learned Rule

event A is a killing by a rival if we can follow a chain of events that connects event A to event B, event B to event E, and event E to an event F that relates two organizations. Events A and E have the same kind of relation, RelationC, to B. All events in the chain are subsets of the same incident D.

1.3.2 Experiments on Synthetic Data

The ease of generating large amounts and data and privacy considerations have led the EELD program to use synthetic data generated by simulators. Next, we describe the results we obtained from simulated data for the CK problem. This data was generated from a run of a Task-Based (TB) simulator developed by Information Extraction and Transport Incorporated (IET). The TB simulator outputs case files, which contain complete and unadulterated descriptions of murder cases. These case files are then filtered for observability, so that facts that would not be accessible to an investigator are eliminated. To make the task more realistic, the simulator output is corrupted, e.g., by misidentifying role players or incorrectly reporting group memberships. This filtered and corrupted data form the evidence files. In the evidence files, facts about each event are represented as ground facts, such as:

```
murder(Murder714)
perpetrator(Murder714, Killer186)
crimeVictim(Murder714, MurderVictim996)
deviceTypeUsed(Murder714, PistolCzech)
```

The synthetic dataset that we used consists of 632 murder events. Each murder event has been labeled as either a positive or negative example of a murder-for-hire. There are 133 positive and 499 negative examples in the dataset. Our task was to learn a theory to correctly classify an unlabeled event as either a positive or negative instance of murder-for-hire. The amount of background knowledge for this dataset is extremely large; consisting of 52 distinct predicate names, and 681,039 background facts in all.

Scaling to large datasets in data mining typically refers to increasing the *number* of training examples that can be processed. Another measure of complexity that is particularly relevant in relational data mining is the *size* of individual examples, i.e. the number of facts used to describe each example. To our knowledge, the challenge problems developed for the EELD program are the largest ILP problems attempted to date in terms of the number of facts in the background knowledge. In order to more effectively process such large examples, we have developed a new ILP method that reduces the number of clauses that are generated and tested.

BETH

The two standard approaches to ILP are bottom-up and top-down [Lavrac & Dzeroski1994]. Bottom-up methods like ALEPH start with a very specific clause (called a *bottom clause*) generated from a seed positive example and generalize it as far as possible without covering negative examples. Top-down methods like FOIL [Quinlan1990] and mFOIL [Lavrac & Dzeroski1994] start with the most general (empty) clause and repeatedly specialize it until it no longer covers negative examples. Both approaches have problems scaling to large examples. When given large amounts of background knowledge, the bottom clause in bottom-up methods becomes intractably large and the increased branching factor in top-down methods greatly impedes their search.

Since top-down and bottom-up approaches have both strengths and weaknesses, we developed a hybrid method that helps reduce search when learning with large amounts of background knowledge. It does not build a bottom clause using a seed example *before* searching for a good clause. Instead, after a random seed example is chosen, it generates literals in a top-down fashion (i.e. guided by heuristic search), except the literals generated are constrained to cover the seed example. Based on this idea, we have developed a system called **B**ottom-clause **E**xploration **T**hrough **H**euristic-search (BETH) in which the bottom clause is not constructed in advance but "discovered" during the search for a good clause. Details of the algorithm are given in [Tang, Mooney, & Melville2003].

Results and Discussion

The performance of ALEPH, mFOIL, and BETH was evaluated using 6-fold cross-validation. The data for each fold was generated by separate runs of the TB simulator. The facts produced by one run of the simulator, only pertain to the entities and relations generated in that run; hence the facts of each fold are unrelated to the others. For each trial, one fold is set aside for testing, while the remaining data is *combined* for training. The total number of Prolog atoms in the data is so large that running more than six folds is not feasible.[2] To test performance on varying amounts of training data, learning curves were generated by testing the system after training on increasing subsets of the overall training data. Note that, for different points on the learning curve, the background knowledge remains the same; only the number of positive and negative training examples given to the system varies.

We compared the three systems with respect to accuracy and training time. Accuracy is defined as the number of correctly classified test cases divided by the total number of test cases. The training time is measured as the CPU time consumed during the training phase. All the experiments were performed on a 1.1 GHz Pentinum with dual processors and 2 GB of RAM. BETH and mFOIL were implemented in Sicstus Prolog version 3.8.5 and ALEPH was implemented in Yap version 4.3.22. Although different Prolog compilers were used, the Yap Prolog compiler has been demonstrated to outperform the Sicstus Prolog compiler, particularly in ILP applications [Santos Costa1999].

The following is a sample rule learned by BETH:

[2]The maximum number of atoms that the Sicstus Prolog compiler can handle is approximately a quarter million.

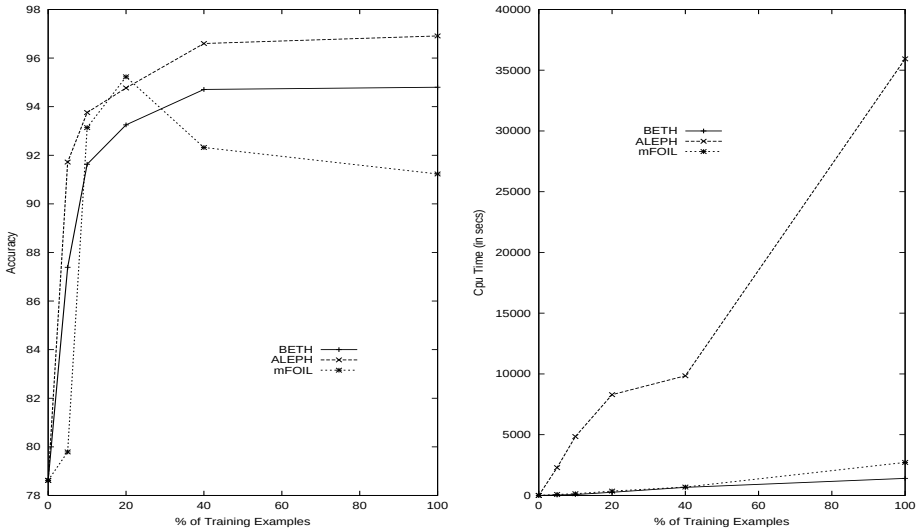


Figure 1.4: Learning Curves for Accuracy and Training Speed

System	*Accuracy*	*CPU Time (mins)*	*# of Clauses*	*Bottom Clause Size*
BETH	94.80% (+/- 2.3%)	23.39 (+/- 4.26)	4483	34
ALEPH	96.91% (+/- 2.8%)	598.92 (+/- 250.00)	63334	4061
mFOIL	91.23% (+/- 4.8%)	45.28 (+/- 5.40)	112904	n/a

Table 1.1: Results on classifying *murder-for-hire* events given all the training data. *# of Clauses* is the total number of clauses tested; and *Bottom Clause Size* is the average number of literals in the bottom clause constructed for each clause in the learned theory. The 90% confidence intervals are given for test *Accuracy* and *CPU time.*

```
murder_for_hire(A):- murder(A), eventOccursAt(A,H),
  geographicalSubRegions(I,H), perpetrator(A,B),
  recipientOfinfo(C,B), senderOfinfo(C,D), socialParticipants(F,D),
  socialParticipants(F,G), payer(E,G), toPossessor(E,D).
```

This rule covered 9 positive examples and 3 negative examples. The rule can be interpreted as: *A* is a murder-for-hire, if *A* is a murder event, which occurs in a city in a subregion of Russia, and in which *B* is the perpetrator, who received information from *D*, who had a meeting with and received some money from *G*.

The results of our experiments are summarized in Figure 1.4. A snapshot of the performance of the three ILP systems given 100% of the training examples is shown in Table 1.1. On the full training set, BETH trains 25 times faster than ALEPH while losing only 2 percentage points in accuracy and it trains twice as fast as mFOIL while gaining 3 percentage points in accuracy. Therefore, we believe that its integration of top-down and bottom-up search is an effective approach to dealing with the problem of

scaling ILP to large examples. The learning curves for training time further illustrate that although BETH and mFOIL appear to scale linearly with the number of training examples, ALEPH's training-time growth is super-linear.

Systems like BETH and ALEPH construct literals based on actual ground atoms in the background knowledge, guaranteeing that the specialized clause covers at least the seed example. On the other hand, mFOIL generates more literals than necessary by enumerating all possible combination of variables. Some such combinations make useless literals; adding any of them to the body of the current clause makes specialized clauses that do not cover any positive examples. Thus, mFOIL wastes CPU time constructing and testing these literals. Since the average predicate arity in the EELD data was small (2), the speedup over mFOIL was not as great, although much larger gains would be expected for data that contains predicates with higher arity.

1.4 Current and Future Research

An under-studied issue in relational data mining is scaling algorithms to very large databases. Most research on ILP and RDM has been conducted in the machine learning and artificial intelligence communities rather than in the database and systems communities. Consequently, there has been insufficient research on systems issues involved in performing RDM in commercial relational-database systems and scaling algorithms to extremely large datasets that will not fit in main memory. Integrating ideas from systems work in data mining and deductive databases [Ramamohanarao & Harland1994] would seem to be critical in addressing these issues.

On the issue of scaling, in addition to the BETH system discussed in section 1.3.2, we are currently working on efficiently learning complex relational concepts from large amounts of data by using stochastic sampling methods. A major shortcoming of ILP is the computational demand that results from the large hypothesis spaces searched. Intelligently sampling these large spaces can provide excellent performance in much less time [Srinivasan1999, Zelezny, Srinivasan, & Page2002].

We are also developing algorithms that learn more robust, probabilistic relational concepts represented as stochastic logic programs [Muggleton2003] and variants. This will enrich the expressiveness and robustness of learned concepts. As an alternative to stochastic logic programs, we are working on learning clauses in a constraint logic programming language where the constraints are Bayesian networks [Page2000, Costa *et al.*2003].

One approach that we plan to investigate further is the use of approximate prior knowledge to induce more accurate, comprehensible relational concepts from fewer training examples [Richards & Mooney1995]. The use of prior knowledge can greatly reduce the burden on users; they can express the "easy" aspects of the task at hand and then collect a small number of training examples to refine and extend this prior knowledge.

We also plan to use active learning to allow our ILP systems to select more effective training examples for interactively learning relational concepts [Muggleton *et al.*1999]. By intelligently choosing the examples for users to label, better extraction accuracy can be obtained from fewer examples, thereby greatly reducing the burden on the users of

our ILP systems.

Another important issue related to data mining for counter-terrorism is privacy preservation. DARPA's counter-terrorism programs have attracted significant public and media attention due to concerns about potential privacy violations (e.g. [Clymer2003]). Consequently, privacy-preserving data mining [Gehrke2002] is another very significant "next generation" issue in data mining.

1.5 Related Work

Although it is the most widely studied, ILP is not the only approach to relational data mining. In particular, other participants in the EELD program are taking alternative RDM approaches to pattern learning for link discovery. This section briefly reviews these other approaches.

1.5.1 Graph-based Relational Learning

Some relational data mining methods are based on learning structural patterns in graphs. In particular, SUBDUE [Cook & Holder1994, Cook & Holder2000] discovers highly repetitive subgraphs in a labeled graph using the minimum description length (MDL) principle. SUBDUE can be used to discover interesting substructures in graphical data as well as to classify and cluster graphs. Discovered patterns do not have to match the data exactly since SUBDUE can employ an inexact graph-matching procedure based on graph edit-distance. SUBDUE has been successfully applied to a number of important RDM problems in molecular biology, geology, and program analysis. It is also currently being applied to discover patterns for link discovery as a part of the EELD project (more details at `http://ailab.uta.edu/eeld/`). Since relational data for LD is easily represented as labeled graphs, graph-based RDM methods like SUBDUE are a natural approach.

1.5.2 Probabilistic Relational Models

Probabilistic relational models (PRM's) [Koller & Pfeffer1998] are an extension of Bayesian networks for handling relational data. Methods for learning Bayesian networks have also been extended to produce algorithms for inducing PRM's from data [Friedman *et al.*1999]. PRM's have the nice property of integrating some of the advantages of both logical and probabilistic approaches to knowledge representation and reasoning. They combine some of the representational expressivity of first-order logic with the uncertain reasoning abilities of Bayesian networks. PRM's have been applied to a number of interesting problems in molecular biology, web-page classification, and analysis of movie data. They are also currently being applied to pattern learning for link discovery as a part of the EELD project.

1.5.3 Relational Feature Construction

One approach to learning from relational data is to first "flatten" or "propositionalize" the data by constructing features that capture some of the relational information and then applying a standard learning algorithm to the resulting feature vectors [Kramer, Lavrač, & Flach2001]. PROXIMITY [Neville & Jensen2000] is a system that constructs features for categorizing entities based on the categories and other properties of other entities to which it is related. It then uses an interactive classification procedure to dynamically update inferences about objects based on earlier inferences about related objects. PROXIMITY has been successfully applied to company and movie data. It is also currently being applied to pattern learning for link discovery as a part of the EELD project.

1.6 Conclusions

Link discovery is an important problem in automatically detecting potential threatening activity from large, heterogeneous data sources. The DARPA EELD program is a U.S. government research project exploring link discovery as an important problem in the development of new counter-terrorism technology. Learning new link-discovery patterns that indicate potentially threatening activity is a difficult data mining problem. It requires discovering novel relational patterns in large amounts of complex relational data. In this work we have shown that ILP methods can extract interesting and useful rules from link-discovery data-bases containing up to hundreds of thousands of items. To do so, we improved search efficiency and computation time per node over current ILP systems.

Most existing data-mining methods assume flat data from a single relational table and are not appropriate for link discovery. Relational data mining techniques, such as inductive logic programming, are needed. Many other problems in molecular biology [Srinivasan *et al.*1996], natural-language understanding [Zelle & Mooney1996], web page classification [Craven *et al.*2000], information extraction [Califf & Mooney1999, Freitag1998], and other areas also require mining multi-relational data. However, relational data mining requires exploring a much larger space of possible patterns and performing complex inference and pattern matching. As a result, current RDM methods are not sufficiently scalable to very large databases. Consequently, we believe that relational data mining is one of the major research topics in the development of the next generation of data mining systems, particularly those in the area of counter-terrorism.

Acknowledgments

This research is sponsored by the Defense Advanced Research Projects Agency and managed by Rome Laboratory under contract F30602-01-2-0571. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency, Rome Laboratory, or the United States Government.

Vítor Santos Costa and Inês de Castro Dutra are on leave from COPPE/Sistemas, Federal University of Rio de Janeiro and were partially supported by CNPq. Many thanks to Hans Chalupksy's group at ISI, in particular to André Valente, who gave us support on using the Task-based simulator. We would like to thank the Biomedical Computing Group support staff and the Condor Team at the Computer Sciences Department of the University of Wisconsin, Madison, for their invaluable help with Condor. We also would like to thank Ashwin Srinivasan for his help with the ALEPH system.

Bibliography

[Breiman1996a] Breiman, L. 1996a. Bagging Predictors. *Machine Learning* 24(2):123–140.

[Breiman1996b] Breiman, L. 1996b. Stacked Regressions. *Machine Learning* 24(1):49–64.

[Califf & Mooney1999] Califf, M. E., and Mooney, R. J. 1999. Relational learning of pattern-match rules for information extraction. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 328–334.

[Clymer2003] Clymer, A. 2003. Pentagon Surveillance Plan Is Described as Less Invasive. *New York Times* May(7).

[Cook & Holder1994] Cook, D. J., and Holder, L. B. 1994. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research* 1:231–255.

[Cook & Holder2000] Cook, D. J., and Holder, L. B. 2000. Graph-based data mining. *IEEE Intelligent Systems* 15(2):32–41.

[Cook & O'Hayon2000] Cook, W., and O'Hayon, G. 2000. Chronology of Russian killings. *Transnational Organized Crime* 4(2).

[Costa *et al.*2003] Costa, V. S.; Page, D.; Qazi, M.; and Cussens, J. 2003. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI-03)*.

[Cowie & Lehnert1996] Cowie, J., and Lehnert, W. 1996. Information extraction. *Communications of the ACM* 39(1):80–91.

[Craven *et al.*2000] Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A. K.; Mitchell, T.; Nigam, K.; and Slattery, S. 2000. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence* 118(1-2):69–113.

[de Castro Dutra *et al.*2002] de Castro Dutra, I.; Page, D.; Costa, V. S.; and Shavlik, J. W. 2002. An empirical evaluation of bagging in inductive logic programming. In *Inductive Logic Programming, 12th International Conference*, volume 2583 of *Lecture Notes in Computer Science*, 48–65. Sydney, Australia: Springer Verlag.

[Dietterich1998] Dietterich, T. G. 1998. Machine-learning research: Four current directions. *The AI Magazine* 18(4):97–136.

[Dutra *et al.*2003] Dutra, I. C.; Page, D.; Santos Costa, V.; Shavlik, J. W.; and Waddell, M. 2003. Towards automatic management of embarassingly parallel applications. In *Proceedings of Europar 2003*, Lecture Notes in Computer Science. Klagenfurt, Austria: Springer Verlag.

[Džeroski & Lavrač2001a] Džeroski, S., and Lavrač, N. 2001a. An introduction to inductive logic programming. In Džeroski, S., and Lavrač, N., eds., *Relational Data Mining*. Berlin: Springer Verlag. 48–73.

[Džeroski & Lavrač2001b] Džeroski, S., and Lavrač, N., eds. 2001b. *Relational Data Mining*. Berlin: Springer Verlag.

[Džeroski2001] Džeroski, S. 2001. Relational data mining applications: An overview. In Džeroski, S., and Lavrač, N., eds., *Relational Data Mining*. Berlin: Springer Verlag. 339–364.

[Freitag1998] Freitag, D. 1998. Information extraction from HTML: Application of a general learning approach. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 517–523. Madison, WI: AAAI Press / The MIT Press.

[Friedman *et al.*1999] Friedman, N.; Getoor, L.; Koller, D.; and Pfeffer, A. 1999. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1300–1307.

[Gehrke2002] Gehrke, J. 2002. Data mining for security and privacy. *SIGKDD Explorations* 4(2):i. Introduction to special issue on Privacy and Security.

[Han & Kamber2001] Han, J., and Kamber, M. 2001. *Data Mining: Concepts and Techniques*. San Francisco: Morgan Kauffmann Publishers.

[Hand, Mannila, & Smyth2001] Hand, D. J.; Mannila, H.; and Smyth, P. 2001. *Principles of Data Mining*. Cambridge, MA: MIT Press.

[Hoche & Wrobel2001] Hoche, S., and Wrobel, S. 2001. Relational learning using constrained confidence-rated boosting. In Rouveirol, C., and Sebag, M., eds., *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, 51–64. Springer-Verlag.

[Houstis *et al.*2000] Houstis, E. N.; Catlin, A. C.; Rice, J. R.; Verykios, V. S.; Ramakrishnan, N.; and Houstis, C. E. 2000. PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software* 26(2):227–253.

[Jensen & Goldberg1998] Jensen, D., and Goldberg, H., eds. 1998. *AAAI Fall Symposium on Artificial Intelligence for Link Analysis*. Menlo Park, CA: AAAI Press.

[Koller & Pfeffer1998] Koller, D., and Pfeffer, A. 1998. Probabilistic frame-based systems. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 580–587. Madison, WI: AAAI Press / The MIT Press.

[Kramer, Lavrač, & Flach2001] Kramer, S.; Lavrač, N.; and Flach, P. 2001. Propositionalization approaches to relational data mining. In Džeroski, S., and Lavrač, N., eds., *Relational Data Mining*. Berlin: Springer Verlag. 262–291.

[Lavrac & Dzeroski1994] Lavrac, N., and Dzeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

[Lehnert & Sundheim1991] Lehnert, W., and Sundheim, B. 1991. A performance evaluation of text-analysis technologies. *AI Magazine* 12(3):81–94.

[McKay, Woessner, & Roule2001] McKay, S. J.; Woessner, P. N.; and Roule, T. J. 2001. Evidence extraction and link discovery (EELD) seedling project, database schema description, version 1.0. Technical Report 2862, Veridian Systems Division.

[Muggleton *et al.*1999] Muggleton, S.; Bryant, C.; Page, C.; and Sternberg, M. 1999. Combining active learning with inductive logic programming to close the loop in machine learning. In Colton, S., ed., *Proceedings of the AISB'99 Symposium on AI and Scientific Creativity (informal proceedings)*.

[Muggleton1992] Muggleton, S. H., ed. 1992. *Inductive Logic Programming*. New York, NY: Academic Press.

[Muggleton2003] Muggleton, S. 2003. Stochastic logic programs. *Journal of Logic Programming*. To appear.

[Neville & Jensen2000] Neville, J., and Jensen, D. 2000. Iterative classification in relational data. In *Papers from the AAAI-00 Workshop on Learning Statistical Models from Relational Data*. Austin, TX: AAAI Press / The MIT Press.

[NIST2003] NIST. 2003. ACE - Automatic Content Extraction. http://www.nist.gov/speech/tests/ace/.

[Page2000] Page, D. 2000. ILP: Just do it! In Lloyd, J.; Dahl, V.; Furbach, U.; Kerber, M.; Lau, K.-K.; Palamidessi, C.; Pereira, L.; Sagiv, Y.; and Stuckey, P., eds., *Proceedings of Computational Logic 2000*, 25–40. Springer Verlag.

[Quinlan1990] Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239–266.

[Quinlan1996] Quinlan, J. R. 1996. Boosting first-order learning. *Algorithmic Learning Theory, 7th International Workshop, Lecture Notes in Computer Science* 1160:143–155.

[Ramamohanarao & Harland1994] Ramamohanarao, K., and Harland, J. 1994. An introduction to deductive database languages and systems. *VLDB Journal* 3:2.

[Richards & Mooney1995] Richards, B. L., and Mooney, R. J. 1995. Automated refinement of first-order Horn-clause domain theories. *Machine Learning* 19(2):95–131.

[Santos Costa, Srinivasan, & Camacho2000] Santos Costa, V.; Srinivasan, A.; and Camacho, R. 2000. A note on two simple transformations for improving the efficiency of an ILP system. In Cussens, J., and Frisch, A., eds., *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, 225–242. Springer-Verlag.

[Santos Costa1999] Santos Costa, V. 1999. Optimising bytecode emulation for Prolog. In *LNCS 1702, Proceedings of PPDP'99*, 261–267. Springer-Verlag.

[Sparrow1991] Sparrow, M. K. 1991. The application of network analysis to criminal intelligence: An assessment of the prospects. *Social Networks* 13:251–274.

[Srinivasan *et al.*1996] Srinivasan, A.; Muggleton, S. H.; Sternberg, M. J.; and King, R. D. 1996. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence* 85:277–300.

[Srinivasan1999] Srinivasan, A. 1999. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery* 3(1):95–123.

[Srinivasan2001] Srinivasan, A. 2001. *The Aleph Manual*. URL: http://oldwww.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph/aleph_toc.html.

[Tang, Mooney, & Melville2003] Tang, L. R.; Mooney, R. J.; and Melville, P. 2003. Scaling up ilp to large examples: Results on link discovery for counter-terrorism. In *submitted to the KDD-03 Workshop on Multi-Relational Data Mining*.

[Wasserman & Faust1994] Wasserman, S., and Faust, K. 1994. *Social Network Analysis: Methods & Applications*. Cambridge, UK: Cambridge University Press.

[Williams & Woessner1995a] Williams, P., and Woessner, P. N. 1995a. Nuclear material trafficking: An interim assessment. *Transnational Organized Crime* 1(2):206–238.

[Williams & Woessner1995b] Williams, P., and Woessner, P. N. 1995b. Nuclear material trafficking: An interim assessment, ridgway viewpoints. Technical Report 3, Ridgway Center, University of Pittsburgh.

[Williams2002] Williams, P. 2002. Patterns, indicators, and warnings in link analysis: The contract killings dataset. Technical Report 2878, Veridian Systems Division.

[Witten & Frank1999] Witten, I. H., and Frank, E. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco: Morgan Kaufmann.

[Woessner1995] Woessner, P. N. 1995. Chronology of nuclear smuggling incidents: July 1991-may 1995. *Transnational Organized Crime* 1(2):288–329.

[Woessner1997] Woessner, P. N. 1997. Chronology of radioactive and nuclear materials smuggling incidents: July 1991-june 1997. *Transnational Organized Crime* 3(1):114–209.

[Wrobel2001] Wrobel, S. 2001. Inductive logic programming for knowledge discovery in databases. In Džeroski, S., and Lavrač, N., eds., *Relational Data Mining*. Berlin: Springer Verlag. 74–101.

[Zelezny, Srinivasan, & Page2002] Zelezny, F.; Srinivasan, A.; and Page, D. 2002. Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th International Conference on Inductive Logic Programming*. Springer Verlag.

[Zelle & Mooney1996] Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the 14th National Conference on Artificial Intelligence*, 1050–1055.

ILP: A Short Look Back and a Longer Look Forward

David Page PAGE@BIOSTAT.WISC.EDU
Dept. of Biostatistics and Medical Informatics
and Dept. of Computer Sciences
University of Wisconsin
1300 University Ave., Rm 5795 Medical Sciences
Madison, WI 53706, USA

Ashwin Srinivasan ASHWIN@COMLAB.OX.AC.UK
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK

Editor: James Cussens and Alan M. Frisch

Abstract

Inductive logic programming (ILP) is built on a foundation laid by research in machine learning and computational logic. Armed with this strong foundation, ILP has been applied to important and interesting problems in the life sciences, engineering and the arts. This paper begins by briefly reviewing some example applications, in order to illustrate the benefits of ILP. In turn, the applications have brought into focus the need for more research into specific topics. We enumerate and elaborate five of these: (1) novel search methods; (2) incorporation of explicit probabilities; (3) incorporation of special-purpose reasoners; (4) parallel execution using commodity components; and (5) enhanced human interaction. It is our hypothesis that progress in each of these areas can greatly improve the contributions that can be made with ILP; and that, with assistance from research workers in other areas, significant progress in each of these areas is possible.

1. Introduction

Inductive logic programming (ILP) has its foundations in computational logic, including logic programming, knowledge representation and reasoning, and automated theorem proving. These foundations go beyond the obvious basis in definite clause logic and SLD-resolution. In addition ILP has utilized such theoretical results from computational logic as Lee's Subsumption Theorem (Lee, 1967), Gottlob's Lemma linking implication and subsumption (Gottlob, 1987), Marcinkowski and Pacholski's result on the undecidability of implication between definite clauses (Marcinkowski and Pacholski, 1992), and many others. In addition to utilizing such theoretical results, ILP depends crucially on important advances in logic programming implementations. For example, many of the applications summarized in the next section were possible only because of fast deductive inference based on indexing, partial compilation, etc. as embodied in the best current Prolog implementations. Finally, research in computational logic has yielded numerous important lessons about knowledge representation in logic that have formed the basis for applications. Just as

one example, definite clause grammars are central to several ILP applications within both natural language processing and bioinformatics.

In his famous address in 1900 to the International Congress of Mathematicians in Paris, David Hilbert commenced thus (from the English translation in (Hilbert, 1902)):

> Who of us would not be glad to lift the veil behind which the future lies hidden; to cast a glance at the next advances of our science ... We know that every age has its own problems, which the following age either solves or casts aside as profitless and replaces by new ones. If we would obtain an idea of the probable development of mathematical knowledge in the immediate future, we must let the unsettled questions pass before our minds and look over the problems which the science of today sets and whose solution we expect from the future.

In a far more humble setting, we present here what we believe to be some pressing issues that have arisen from the most challenging ILP applications of today. These are:

1. The development of novel search methods;
2. Techniques for incorporating explicit probabilities into ILP;
3. The use of special-purpose reasoners in ILP;
4. Techniques for parallel execution using commodity components; and
5. Enhancing human-computer interaction to make ILP systems true collaborators with human experts.

It is our belief that adequate solutions to the concomitant problems posed by these issues will greatly improve the quality and type of assistance that can be rendered by ILP systems. Further, we fully expect such solutions are obtainable in the future, with the assistance of research workers from machine learning, algorithm development, computational logic, and experimental psychology.

The rest of the paper is organised as follows. Section 2 gives a brief review of the some the application areas that have motivated the research issues enumerated above. Each of these issues is examined in greater detail in Sections 3–7. Section 8 concludes the paper.

2. Challenging Application Areas for ILP

One of the most important application domains for machine learning in general is bioinformatics, broadly interpreted. This domain is particularly attractive for (1) its obvious importance to society, and (2) the plethora of large and growing data sets. Data sets obviously include the newly completed and available DNA sequences for *C. elegans* (nematode), *Drosophila* (fruitfly), and (depending on one's definitions of "completed" and "available") man. But other data sets include gene expression data (recording the degree to which various genes are expressed as protein in a tissue sample), bio-activity data on potential drug molecules, x-ray crystallography and NMR data on protein structure, and data from novel techniques in proteomics. Applications within bioinformatics include protein structure prediction (Muggleton et al., 1992, Turcotte et al., 1998), mutagenicity prediction (King et al., 1996), and pharmacophore discovery (Marchand-Geneste et al., 2002, Finn et al., 1998)

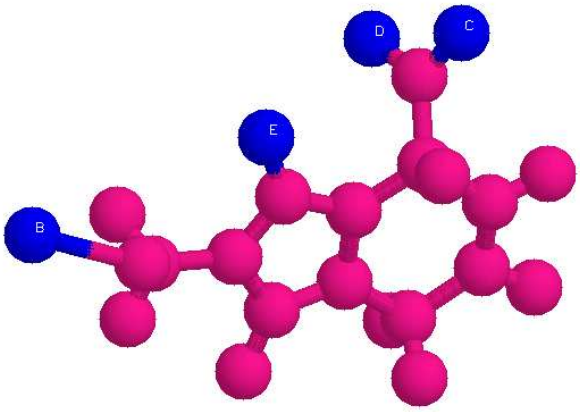


Figure 1: ACE inhibitor number 1 with highlighted 4-point pharmacophore.

(discovery of a 3D substructure responsible for drug activity that can be used to guide the search for new drugs with similar activity). ILP is particularly well-suited for bioinformatics tasks because of its abilities to take into account background knowledge and work directly with structured data. For example, the following is a potential pharmacophore for ACE inhibition (a form of hypertension medication), where the spatial relationships are described through pairwise distances.[1]

```
Molecule A is an ACE inhibitor if:
   molecule A contains a zinc binding site B, and
   molecule A contains a hydrogen acceptor C, and
   the distance between B and C is 7.9 +/- .75 Angstroms, and
   molecule A contains a hydrogen acceptor D, and
   the distance between B and D is 8.5 +/- .75 Angstroms, and
   the distance between C and D is 2.1 +/- .75 Angstroms, and
   molecule A contains a hydrogen acceptor E, and
   the distance between B and E is 4.9 +/- .75 Angstroms, and
   the distance between C and E is 3.1 +/- .75 Angstroms, and
   the distance between D and E is 3.8 +/- .75 Angstroms.
```

Figures 1 and 2 show two different ACE inhibitors with the parts of the pharmacophore highlighted and labeled. The preceding rule was automatically translated directly from logic. It illustrates another strength of ILP, in that logical rules are easily translated into

1. Hydrogen acceptors are atoms with a weak negative charge. Ordinarily, zinc-binding would be irrelevant; it is relevant here because ACE is one of several proteins in the body that typically contains an associated zinc ion. The error tolerance on distances was fixed to 0.75 Angstroms based on the recommendation of a domain expert; multiple possible error tolerances can be incorporated into the search if that is desired instead. 1.0 and 0.75 are typical tolerances preferred by chemists.

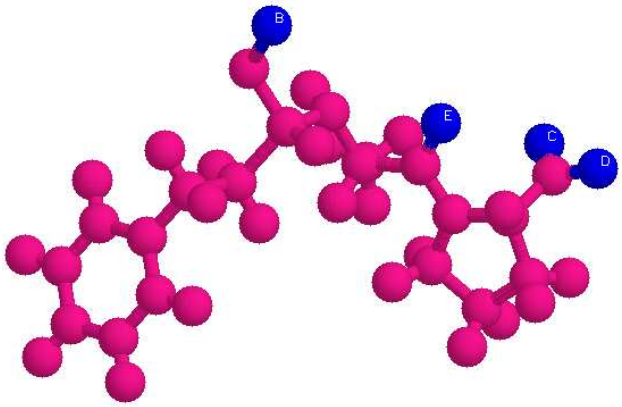


Figure 2: ACE inhibitor number 2 with highlighted 4-point pharmacophore.

English. If the vocabulary for the rules is meaningful to the domain experts, as in this case, then ILP discoveries are directly comprehensible to humans, at least to domain experts.

Note that the features (acceptors and zinc-binding) are related to one another by distances, and a typical molecule may have many atoms or groups that potentially could play the role of a given feature in the rule. Hence just testing whether a molecule satisfies a rule is itself a constraint-satisfaction problem, where features are variables and distances are constraints. For this reason, ordinary feature based learners (e.g., decision tree algorithms), cannot learn rules of this form unless each possible rule is itself encoded as a single feature. But the problem with that approach is that it leads to millions of features for a typical set of molecules.

A very different type of domain for machine learning is natural language processing (NLP). This domain also includes a wide variety of tasks such as part-of-speech tagging, grammar learning, information retrieval, and information extraction. Arguably, natural language translation (at least, very rough-cut translation) is now a reality—witness for example the widespread use of Altavista's Babelfish (http://babel.altavista.com/). Machine learning techniques are aiding in the construction of information extraction engines that fill database entries from document abstracts or web pages (e.g., (Craven and Kumlien, 1999)). NLP became a major application focus for ILP in particular with the ESPRIT project ILP2. A strength of ILP for NLP is that grammars can be represented as logic programs, so the same algorithms used to learn pharmacophores can be applied to learning grammars or portions of grammars.

A third popular and challenging application area for machine learning is knowledge discovery from large databases with rich data formats, which might contain for example satellite images, audio recordings, movie files, etc. While Dzeroski has shown how ILP

applies very naturally to knowledge discovery from ordinary relational databases (Dzeroski, 1996), advances are needed to deal with multimedia databases.

ILP has advantages over other machine learning techniques for all of the preceding application areas. Nevertheless, these applications also highlight the following shortcomings of present ILP technology:

- Techniques such as bigrams and trigrams, or the more complex hidden Markov models, Bayes nets and dynamic Bayes nets, can expressly represent the probabilities inherent in tasks such as part-of-speech tagging, alignment of proteins, robot maneuvering, etc. Few ILP systems need to have such capabilities.
- ILP systems have higher time and space requirements than other machine learning systems, making it difficult to apply them to large data sets. Novel search algorithms and parallel processing need to be explored.
- ILP works well when data and background knowledge are cleanly expressible in first-order logic. But what can be done when databases contain images, audio, movies, etc.? ILP needs to learn lessons from constraint logic programming regarding the incorporation of special-purpose techniques for handling special data formats.
- In scientific knowledge discovery, for example in the domain of bioinformatics, it would be beneficial if ILP systems could collaborate with scientists rather than merely running in batch mode. If ILP does not take this step, other forms of collaborative scientific assistants will be developed, supplanting ILP's position within these domains.

The directions for further research that are discussed in the following sections address these shortcomings, in the same order.

3. Improved/Novel Search Methods

Most ILP algorithms search a lattice of clauses ordered by subsumption. They seek a clause that maximizes some function of the size of the clause and coverage of the clause, i.e. the numbers of positive and negative examples entailed by the clause together with the background theory. Depending upon how they search this lattice, these ILP algorithms are classified as either specific-to-general (based on least general generalization) or general-to-specific (based on refinement). Algorithms are further classified by whether they perform a greedy search, beam search, admissible search, etc. But a large space of possible algorithms still remains unexplored—algorithms that are neither top-down nor bottom-up, nor even necessarily deterministic. For other challenging logic or artificial intelligence tasks outside ILP, great progress has been made in the development of novel search stategies. The best-known case is satisfiability, where GSAT (Selman et al., 1992) made a substantial improvement over Davis-Putnam, then WalkSAT (Selman et al., 1994) improved upon GSAT, and where more recently novel versions of Davis-Putnam with rapid random restarts have outperformed WalkSAT (Gomes et al., 2000). Consequently, a promising research direction is to apply novel search strategies such as these to ILP.

ILP algorithms face not one but two difficult search problems. In addition to the search of the lattice of clauses, already described, simply testing the coverage of a clause involves

repeated searches for proofs—"if I assume this clause is true, does a proof exist for that example?" Some work on stochastic search in ILP already has been done, and it addressed this latter search problem. Sebag and Rouveirol (Sebag and Rouveirol, 1997) employed stochastic matching, or theorem proving, and obtained efficiency improvements over Progol in the prediction of mutagenicity, without sacrificing predictive accuracy or comprehensibility. More recently, Botta, Giordana, Saitta, and Sebag have pursued this approach further, continuing to show the benefits of replacing deterministic matching with stochastic matching (Giordana et al., 2000). But at the center of ILP is the search of the clause lattice. Genetic algorithms have been employed for searching this lattice, but more work on novel search strategies is needed. The remainder of this section briefly outlines several directions for such research.

First, one can easily imagine variants of GSAT and WalkSAT tailored to search a lattice of clauses instead of truth assignments, trying to maximize consistency with a data set rather than clauses satisfied in a Boolean CNF formula. A natural ILP variant of GSAT performs as follows. It first draws a random first-order definite clause, rather than a random truth assignment. Instead of "flipping" the truth assignments of individual variables, its moves involve adding or deleting literals in the clause. The ILP variant of WalkSAT is a very similar algorithm. The difference is that with some probability p the algorithm makes a random move—it randomly selects an efficacious literal to add or delete. An efficacious addition is a literal that, when added, will cause the clause not to cover some negative example; an efficacious deletion is a literal that, when deleted, will permit the clause to cover a previously-uncovered positive example.

We have conducted preliminary experiments using an implementation of these algorithms within the Aleph system. On an artificial domain consisting of random graphs, we find that ILP-WSAT outperforms ILP-GSAT. Both algorithms perform better than a routine greedy search, and find useful clauses in cases where it is intractable to use a complete search.

These results are promising, but much more more research can be done. First, the procedures described still search for one clause at a time. To learn multiple clauses, they employ the standard greedy-covering heuristic. Can stochastic searches be formulated that search the space of entire *theories* rather than clauses? Second, in GSAT or WSAT scoring a given truth assignment is very fast. In contrast, scoring a clause can be much more time consuming because it involves repeated theorem proving. Therefore, it might be beneficial to combine the ILP GSAT and WSAT algorithms with the stochastic theorem proving mentioned earlier. Third, the number of literals that can be built from a language often is infinite, so we cannot test all possible additions of a literal. Our approach has been to base any given iteration of the algorithm on a "bottom clause" built from a "seed example," based on the manner in which the ILP system Progol (Muggleton, 1995) constrains its search space. Fourth, other types of stochastic search could be tried, such as simulated annealing.

We have noted already that Davis-Putnam has been improved substantially through alternative settings of parameters and *rapid random restarts* (RRR) (Gomes et al., 2000). This success suggests that RRR might also be used in ILP to improve refinement-based searches. A start in this direction has been made very recently, and the results indicate that refinement-based search with RRR is indeed a promising approach Zelezny et al. (2002).

Much more research is needed to determine appropriate parameters for such a search, including how rapidly the restarts should occur.

4. Probabilistic Inference: ILP and Bayes Nets

Bayesian networks have largely supplanted traditional rule-based expert systems. Why? Because in task after task artificial intelligence practitioners have realized that probabilities are central. For example, in medical diagnosis few universally true rules exist and few entirely accurate laboratory experiments are available. Instead, probabilities are needed to model the task's inherent uncertainty. Bayes nets are designed specifically to model probability distributions and to reason about these distributions accurately and (in some cases) efficiently. Consequently, in many tasks including medical diagnosis (Heckerman et al., 1992), Bayes nets have been found to be superior to rule-based systems. Interestingly, inductive inference, or machine learning, has turned out to be a very significant component of Bayes net reasoning. Inductive inference from data is particularly important for developing or adjusting the conditional probability tables for various network nodes, but also is used in some cases even for developing or modifying the structure of the network itself.

In spite of these advantages, a Bayes net is less expressive than first-order logic, on a par with propositional logic instead. Consequently, while a Bayes net is a graphical representation, it cannot represent relational structures. The only relationships captured by the graphs are conditional dependencies among variables. This failure to capture other relational information is particularly troublesome when using the Bayes net representation in learning. For a concrete illustration, consider the task of pharmacophore discovery. It would be desirable to learn probabilistic predictors, e.g., what is the probability that a given structural change to the molecule fluoxetine (Prozac) will yield an equally effective anti-depressant (specifically, serotonin reuptake inhibitor)? To build such a probabilistic predictor, we might choose to learn a Bayes net from data on serotonin reuptake inhibitors. Unfortunately, while a Bayes net can capture the probabilistic information, it cannot capture the structural properties of a molecule that are predictive of biological activity.

The inability of Bayes nets to capture relational structure is well known and has led to the recent extension to *probabilistic relational models* (PRMs) and the study of learning algorithms for such models (Getoor et al., 2001). Probabilistic relational models are an extension of Bayes nets to multiple relational tables, as in a relational database. Because so many real-world data mining tasks are relational in nature, and hence require multiple relational tables, the power of PRMs has immediately been widely recognized. It is worth bearing in mind, nevertheless, that PRMs fall short of the expressivity of first-order logic, or even of Datalog, and that the learning algorithms are very different from those employed within ILP. An interesting alternative for ILP researchers to examine is learning clauses with probabilities attached. It will be important in particular to examine how such representations and learning algorithms compare with PRMs and PRM learning algorithms. It may well be the case that these closely related research directions can benefit greatly from one another. Several candidate probabilistic logic representations have been proposed and include probabilistic logic programs, Bayesian logic programs, stochastic logic programs, and probabilistic constraint logic programs; Cussens provides a nice survey of these representations (Cussens, 1999). Study already has begun into algorithms and applications for

learning stochastic logic programs (Muggleton, 2000) and Bayesian logic programs (Kersting and Raedt, 2001), and these are exciting areas for further work. The first-order representations closest to Bayes nets are the representation of Ngo and Haddawy (Ngo and Haddawy, 1997, 1995) and the logic programs of Kersting and De Raedt (Kersting et al., 2000). The remainder of this section points to approaches for, and potential benefits of, learning clauses in the representation of Ngo and Haddawy or a similar representation.

Clauses in the representation of Ngo and Haddawy may contain random variables as well as ordinary logical variables. A clause may contain at most one random variable in any one literal, and random variables may appear in body literals only if a random variable appears in the head. Finally, such a clause also has a Bayes net fragment attached, which may be thought of as a constraint. This fragment has a very specific form. It is a directed graph of node depth two (edge depth one), with all the random variables from the clause body as parents of the random variable from the clause head.[2] Figure 3 provides an example of such a clause as might be learned in pharmacophore discovery (conditional probability table not shown). This clause enables us to specify, through a conditional probability table, how the probability of a molecule being active depends on the particular values assigned to the distance variables $D1$, $D2$, and $D3$. In general, the role of the added constraint in the form of a Bayes net fragment is to define a conditional probability distribution over the random variable in the head, conditional on the values of the random variables in the body. When multiple such clauses are chained together during inference, a larger Bayes net is formed that defines a joint probability distribution over the random variables.

We conjecture that existing ILP algorithms can effectively learn clauses of this form with the following modification. For each clause constructed by the ILP algorithm, collect the positive examples covered by the clause. Each positive example provides a value for the random variable in the head of the clause, and because the example is covered, the example together with the background knowledge provides values for the random variables in the body. These values, over all the covered positive examples, can be used as the data for constructing the conditional probability table (conditional probability table) that accompanies the attached Bayes net fragment. When all the random variables are discrete, a simple, standard method exists for constructing conditional probability tables from such data and is described nicely in (Heckerman, 1995). If some or all of the random variables are continuous, then under certain assumptions again simple, standard methods exist. For example, under one set of assumptions linear regression can be used, and under another naive Bayes can be used. In fact, the work by Srinivasan and Camacho (Srinivasan and Camacho, 1999) on predicting levels of mutagenicity and the work by Craven and colleagues (Craven and Slattery, 1998, Craven and Kumlien, 1999) on information extraction can be seen as special cases of this proposed approach, employing linear regression and naive Bayes, respectively.

While the approach just outlined appears promising, of course it is not the only possible approach and may not turn out to be the best. More generally, ILP and Bayes net learning are largely orthogonal. The former handles relational domains well, while the latter handles probabilities well. And both Bayes nets and ILP have been applied successfully to a variety

2. This is not exactly the definition provided by Ngo and Haddawy, but it is an equivalent one. Readers interested in deductive inference with this representation are encouraged to see (Ngo and Haddawy, 1997, 1995).

```
drug(Molecule,Activity_Level):-
    contains_hydrophobe(Molecule,Hydrophobe),
    contains_basic_nitrogen(Molecule,Nitrogen),
    contains_hydrogen_acceptor(Molecule,Acceptor),
    distance(Molecule,Hydrophobe,Nitrogen,D1),
    distance(Molecule,Hydrophobe,Acceptor,D2),
    distance(Molecule,Nitrogen,Acceptor,D3).
```

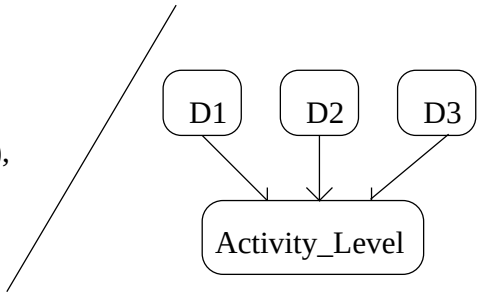


Figure 3: A clause with a Bayes net fragment attached (*conditional probability table* not included). The random variables are *Activity_Level*, *D1*, *D2*, and *D3*. Rather than using a hard range in which the values of *D1*, *D2*, and *D3* must fall, as the pharmacophores described earlier, this new representation allows us to describe a probability distribution over *Activity_Level* in terms of the values of *D1*, *D2*, and *D3*. For example, we might assign higher probabilities to high *Activity_Level* as *D1* gets closer to 3 Angstroms from either above or below. The conditional probability table itself might be a linear regression model, i.e. a linear function of *D1*, *D2*, and *D3* with some fixed variance assumed, or it might be a discretized model, or other.

of tasks. Therefore, it is reasonable to hypothesize the existence and utility of a representation and learning algorithms that effectively capture the advantages of both Bayes net learning and ILP. The space of such representations and algorithms is large, so combining Bayes net learning and ILP is an area of research that is not only promising but also wide open for further work.

5. Special-purpose Reasoning Mechanisms

One of the well-documented success stories of computational logic is constraint logic programming. And one of the reasons for this success is the ability to integrate logic and special purpose reasoners or constraint solvers. Many ILP applications could benefit from the incorporation of special-purpose reasoning mechanisms. Indeed, the approach advocated in Section 3.1 to incorporating probabilities in ILP can be thought of as invoking special purpose reasoners to construct constraints in the form of Bayes net fragments. The work by Srinivasan and Camacho mentioned there uses linear regression to construct a constraint, while the work by Craven and Slattery uses naive Bayes techniques to construct a constraint. The point that is crucial to notice is that ILP requires a "constraint constructor," such as linear regression, in addition to the constraint solver required during deduction. Let's now turn to consideration of tasks where other types of constraint generators might be useful.

Consider the general area of knowledge discovery from databases. Suppose we take the standard logical interpretation of a database, where each relation is a predicate, and each tuple in the relation is a ground atomic formula built from that predicate. Džeroski and

Lavrač show how ILP techniques are very naturally suited to this task, if we have an "ordinary" relational database Džeroski and Lavrač (2001). But now suppose the database contains some form of complex objects, such as images. Simple logical similarities may not capture the important common features across a set of images. Instead, special-purpose image processing techniques may be required, such as those described by Leung and colleagues (Leung and Malik, 2001, Leung et al., 1998). In addition to simple images, special-purpose constraint constructors might be required when applying ILP to movie (e.g. MPEG) or audio (e.g. MP3) data, or other data forms that are becoming ever more commonplace with the growth of multimedia. For example, a fan of the Bach, Mozart, and Brian Wilson would love to be able to enter her/his favorite pieces, have ILP with a constraint generator build rules to describe these favorites, and have the rules suggest other pieces or composers s/he should access. As multimedia data becomes more commonplace, ILP can remain applicable only if it is able to incorporate special- purpose constraint generators.

Alan Frisch and David Page have shown that the ordinary subsumption ordering over formulas scales up quite naturally to incorporate constraints (Frisch and Page, 1995). Nevertheless, that work does not address some of the hardest issues, such as how to ensure the efficiency of inductive learning systems based on this ordering and how to design the right types of constraint generators. These questions require much further research involving real-world applications such as multimedia databases.

One final point about special purpose reasoners in ILP is worth making. Constructing a constraint may be thought of as inventing a predicate. Predicate invention within ILP has a long history (Muggleton and Buntine, 1988, Wirth and O'Rorke, 1991, Zelle and Mooney, 1993, Muggleton, 1994). General techniques for predicate invention encounter the problem that the space of "inventable" predicates is unconstrained, and hence allowing predicate invention is roughly equivalent to removing all bias from inductive learning. While removing bias may sound at first to be a good idea, inductive learning in fact requires bias (Mitchell, 1980, 1982). Special purpose techniques for constraint construction appear to make it possible to perform predicate invention in way that is limited enough to be effective (Srinivasan and Camacho, 1999, Craven and Kumlien, 1999).

6. Parallel Execution

Although ILP has numerous advantages over other types of machine learning, including advantages mentioned at the start of the previous section, it has two particularly notable disadvantages—excessive run time and space requirements. Fortunately for ILP, at the same time that larger applications are highlighting these disadvantages, parallel processing "on the cheap" is becoming widespread. Most notable is the widespread use of "Beowulf clusters" (Becker et al., 1995) and of "Condor pools" (Litzkow et al., 1988), arrangements that connect tens, hundreds, or even thousands of personal computers or workstations to permit parallel processing. Admittedly, parallel processing cannot change the order of the time or space complexity of an algorithm. But most ILP systems already use broad constraints, such as maximum clause size, to hold down exponential terms. Rather, the need is to beat back the large constants brought in by large real-world applications.

Yu Wang and David Skillicorn recently developed a parallel implementation of Progol under the Bulk Synchronous Parallel model and claim superlinear speedup from this im-

plementation (Skillicorn and Wang, 2001). The remainder of this section describes how large-scale parallelism can be achieved very simply in a complete general-to-specific search ILP algorithm. From this discussion, one can imagine more interesting approaches for other types of general-to-specific such as greedy search.

The ideal in parallel processing is a decrease in processing time that is a linear function, with a slope near 1, of the number of processors used. (In some rare cases it is possible to achieve superlinear speed-up.) The barriers to achieving the ideal are (1) overhead in communication between processes and (2) competition for resources between processes. Therefore, a good parallel scheme is one where the processes are relatively independent of one another and hence require little communication or resource sharing. The key observation in the design of the parallel ILP scheme is that two competing hypotheses can be tested against the data completely independently of one another. Therefore the approach advocated here is to distribute the hypothesis space among different processors for testing against the data. These processors need not communicate with one another during testing, and they need not write to a shared memory space.

In more detail, for complete search a parallel ILP scheme can employ a master-worker design, where the master assigns different segments of the hypothesis space to workers that then test hypotheses against the data. Workers communicate back to the master all hypotheses achieving a pre-selected minimum valuation score (e.g. 95 % accuracy) on the data. As workers become free, the master continues to assign new segments of the space until the entire space has been explored. The only architectural requirements for this approach are (1) a mechanism for communication between the master and each worker and (2) read access for each worker to the data. Because data do not change during a run, this scheme can easily operate under either a shared memory or message passing architecture; in the latter, we incur a one-time overhead cost of initially communicating the data to each worker. The only remaining overhead, on either architecture, consists of the time spent by the master and time for master-worker communication. In "needle in a haystack" domains, which are the motivation for complete search, one expects very few hypotheses to be communicated from workers to the master, so overhead for the communication of results will be low. If it also is possible for the master to rapidly segment the hypothesis space in such a way that the segments can be communicated to the workers succinctly, then overall overhead will be low and the ideal of linear speed-up can be realized. One implementation of this approach has, in fact, already been tested on the pharmacophore discovery task mentioned in the introduction (Kamal et al., 2001).

Undoubtedly there are a variety of other complete search, or exact, parallel schemes that can be implemented, and the investigation of such schemes is a key area for further research. For all such schemes, two crucial questions should be answered. First, under what conditions is the new parallel scheme faster than existing ones? Second, are the solutions returned by the parallel complete search significantly better than those returned by a stochastic search? This second question brings us back to our second research direction, that of stochastic ILP algorithms. Of course, it is not necessary to choose between stochastic and parallel algorithms. The stochastic algorithms proposed earlier can themselves be implemented on a parallel processor, in the simplest case by replacing restarts by independent searches running at the same time. How will the results of parallel stochastic searches compare with

those of parallel complete searches, if both searches are provided with the same number of processors and the same amount of time?

7. Interaction with Human Experts

To discover new knowledge from data in fields such as telecommunications, molecular biology, or pharmaceuticals, it would be beneficial if a machine learning system and a human expert could act as a team, taking advantage of the computer's speed and the expert's knowledge and skills. ILP systems have three properties that make them natural candidates for collaborators with humans in knowledge discovery:

Declarative Background Knowledge ILP systems can make use of declarative background knowledge about a domain in order to construct hypotheses. Thus a collaboration can begin with a domain expert providing the learning system with general knowledge that might be useful in the construction of hypotheses. Most ILP systems also permit the expert to define the hypothesis space using additional background knowledge, in the form of a *declarative bias*.

Natural descriptions of structured examples Feature-based learning systems require the user to begin by creating features to describe the examples. Because many knowledge discovery tasks involve complex structured examples, such as molecules, users are forced to choose only composite features such as molecular weight—thereby losing information—or to invest substantial effort in building features that can capture structure (see (Srinivasan et al., 1996) for a discussion in the context of molecules). ILP systems allow a structured example to be described naturally in terms of the objects that compose it, together with relations among those objects. The 2-dimensional structure of a molecule can be represented directly using its atoms as the objects and bonds as the relations; 3-dimensional structure can be captured by adding distance relations.

Human-Comprehensible Output ILP systems share with propositional–logic learners the ability to present a user with declarative, comprehensible rules as output. Some ILP systems can return rules in English along with visual aids. For example, the pharmacophore description and corresponding figures in Section 2 were generated automatically by Progol.

Despite the useful properties just outlined, ILP systems—like other machine learning systems—have a number of shortcomings as collaborators with humans in knowledge discovery. One shortcoming is that most ILP systems return a single theory based on heuristics, thus casting away many clauses that might be interesting to a domain expert. But the only currently existing alternative is the version space approach, which has unpalatable properties that include inefficiency, poor noise tolerance, and a propensity to overwhelm users with too large a space of possible hypotheses. Second, ILP systems cannot respond to a human expert's questions in the way a human collaborator would. They operate in simple batch mode, taking a data set as input, and returning a hypothesis on a take-it-or-leave-it basis. Third, ILP systems do not question the input data in the way a human collaborator would, spotting surprising (and hence possibly erroneous) data points and raising questions about

them. Some ILP systems will flag mutually inconsistent data points but to our knowledge none goes beyond this. Fourth, while a human expert can provide knowledge-rich forms of hypothesis justification, for example relating a new hypothesis to existing beliefs, ILP systems merely provide accuracy estimates as the sole justification.

To build upon ILP's strengths as a technology for human-computer collaboration in knowledge discovery, the above shortcomings should be addressed. ILP systems should be extended to display the following capabilities.

1. Maintain and summarize alternative hypotheses that explain or describe the data, rather than providing a single answer based on a general-purpose heuristic.

2. Propose to human experts practical sequences of experiments to refine or distinguish between competing hypotheses.

3. Provide non-numerical justification for hypotheses, such as relating them to prior beliefs or illustrative examples (in addition to providing numerical accuracy estimates).

4. Answer an expert's questions regarding hypotheses.

5. Consult the expert regarding anomalies or surprises in the data.

In fact, the other four directions already discussed in this paper already go a long way toward capabilities 1 and 3. Both stochastic search and parallel search technologies make it possible to find a potentially large number of alternative hypotheses more efficiently, thus helping to provide capability 1. The ability to provide probability distributions over the bindings for variables in competing hypotheses can potentially provide much more information about these hypotheses. This additional information can be useful in justifying one hypothesis over another. As human experts look more closely at hypotheses, and ask more details about how they fit the data (beyond a simple accuracy number), again these probability distributions can provide further insight.

Work relevant to capability 2 for ILP has been done recently with a goal very different from human-computer interaction. Bryant and colleagues have developed a system to automatically propose *and execute* experiments related to yeast metabolism Bryant et al. (2001). The system contains an ILP system interfaced with a robot to perform experiments. Each experiment tests whether a particular "knock-out" strain of yeast will grow on a particular medium. The "knock-outs" are variants of yeast, each with a single gene altered so that its functionality is lost to the organism. The goal of this work is to automatically induce a logical model for a portion of yeast metabolism. Although the spirit of this work is virtually the opposite of human-computer interaction, the approach to experiment proposal is relevant for human-computer interaction.

Addressing human-computer interface issues obviously requires a variety of logical and artificial intelligence expertise. Thus contributions from other areas of artificial intelligence and computational logic, such as the study of logical agents, will be vital.

8. Conclusions

ILP has attracted great interest within the machine learning and artificial intelligence communities at large because of its logical foundations, its ability to utilize background knowl-

edge and structured data representations, and its comprehensible results. But most of all, the interest has come from ILP's application successes. Nevertheless, ILP needs further advances to maintain this record of success, and these advances require further contributions from other areas of computational logic. System builders and parallel implementation experts are needed if the ILP systems of the next decade are to scale up to the next generation of data sets, such as those being produced by Affymetrix's gene expression microarrays and Celera's shotgun approach to DNA sequencing. Research workers on probability and logic are required if ILP is to avoid being supplanted by the next generation of extended Bayes net learning systems. Experts on constraint satisfaction and constraint logic programming have the skills necessary to bring successful stochastic search techniques to ILP and to allow ILP techniques to extend to multimedia databases. Paraphrasing the closing words of Hilbert in his 1900 address: that ILP may completely fulfil its high mission, may the next decade bring gifted masters and many zealous and enthusiastic disciples!

Acknowledgements

The first author was supported in part by NSF grant 9987841 and by grants from the University of Wisconsin Graduate School and Medical School.

References

D. Becker, T. Sterling, D. Savarese, E. Dorband, U. Ranawake, and C. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, pages 11–14, 1995.

C. Bryant, S. Muggleton, S. Oliver, D Kell, P. Reiser, and R. King. Combining inductive logic programming, active learning and robotics to discover the function of genes. *Electronic Transactions in Artificial Intelligence*, 5-B1(012):1–36, 2001.

M. Craven and J. Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 77–86, Heidelberg, Germany, 1999. AAAI Press.

M. Craven and S. Slattery. Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of the Eighth International Conference on Inductive Logic Programming (ILP-98)*, pages 38–52. Springer Verlag, 1998.

J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 126–133. Stockholm, Sweden, 1999.

S. Džeroski and N. Lavrač. An introduction to inductive logic programming. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 48–71. Springer, Berlin, 2001.

S. Dzeroski. Inductive logic programming and knowledge discovery in databases. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. 1996.

P. Finn, S. Muggleton, D. Page, and A. Srinivasan. Discovery of pharmacophores using Inductive Logic Programming. *Machine Learning*, 30:241–270, 1998.

A. M. Frisch and C. D. Page. Building theories into instantiation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, chapter 13, pages 307–335. Springer, Berlin, 2001.

A. Giordana, L. Saitta, M. Sebag, and M. Botta. Analyzing relational learning in the phase transition framework. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 311–318, Stanford, 2000. Morgan Kaufmann.

C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.

G. Gottlob. Subsumption and implication. *Information Processing Letters*, 24(2):109–111, 1987.

D. Heckerman. A tutorial on learning with bayesian networks. Microsoft Technical Report MSR-TR-95-06, 1995.

D. Heckerman, E. Horvitz, and B. Nathwani. Toward normative expert systems, part I: The pathfinder project. *Methods of Information in Medicine*, 31:90–105, 1992.

D. Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8: 437–479, 1902. English translation of original German, provided by Mary Winston.

A.H. Kamal, J. Graham, and C.D. Page. An approach to parallel data mining for pharmacophore discovery. In *Proceedings of the Tenth International Conference on Intelligent Systems*, pages 100–103, Washington, D.C., June 2001.

K. Kersting and L. De Raedt. Towards combining inductive logic programming and bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming*, pages 118–137. Berlin: Springer LNAI 2157, 2001.

K. Kersting, L. De Raedt, and S. Kramer. Interpreting bayesian logic programs. In L. Getoor and D. Jensen, editors, *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, Austin, Texas, 2000. AAAI Press, Technical Report WS-00-06.

R. King, S. Muggleton, A. Srinivasan, and M. Sternberg. Structure-activity relationships derived by machine learning: the use of atoms and their bond connectives to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93:438–442, 1996.

C. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms*. PhD thesis, University of California, Berkeley, 1967.

T. Leung, M. Burl, and P. Perona. Probabilistic affine invariants for recognition. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

M. Litzkow, M. Livny, and M. Mutka. Condor—a hunter of idle workstations. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 104–111, 1988.

N. Marchand-Geneste, K. Watson, B. Alsberg, and R. King. A new approach to pharmacophore mapping and qsar analysis using inductive logic programming. application to thermolysin inhibitors and glycogen phosphorylase b inhibitors. *Journal of Medicinal Chemistry*, 45(2):399–409, January 2002.

J. Marcinkowski and L. Pacholski. Undecidability of the horn-clause implication problem. In *Proceedings of the 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 354–362. IEEE, 1992.

T.M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, 1980.

T.M. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.

S. Muggleton. Predicate invention and utilization. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):127–130, 1994.

S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

S. Muggleton. Learning stochastic logic programs. In *Proceedings of the AAAI2000 Workshop on Learning Statistical Models from Relational Data*. AAAI, 2000.

S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657, 1992.

L. Ngo and P. Haddawy. Probabilistic logic programming and bayesian networks. *Algorithms, Concurrency, and Knowledge: LNCS 1023*, pages 286–300, 1995.

L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.

M. Sebag and C. Rouveirol. Tractable induction and classification in FOL. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–892. Nagoya, Japan, 1997.

B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI Press, 1994.

B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446. AAAI Press, 1992.

D. Skillicorn and Y. Wang. Parallel and sequential algorithms for data mining using inductive logic. *Knowledge and Information Systems*, 3(4):405–421, 2001.

A. Srinivasan and R.C. Camacho. Numerical reasoning with an ILP system capable of lazy evaluation and customised search. *Journal of Logic Programming*, 40:185–214, 1999.

A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Intelligence*, 85(1,2):277–299, 1996.

M. Turcotte, S. Muggleton, and M. Sternberg. Application of inductive logic programming to discover rules governing the three-dimensional topology of protein structures. In *Proceedings of the Eighth International Conference on Inductive Logic Programming (ILP-98)*, pages 53–64. Springer Verlag, 1998.

R. Wirth and P. O'Rorke. Constraints on predicate invention. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 457–461. Kaufmann, 1991.

F. Zelezny, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In S. Matwin, editor, *Proceedings of the Twelfth International Conference on Inductive Logic Programming*. Springer-Verlag, 2002.

J. Zelle and R. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 817–822, San Mateo, CA, 1993. Morgan Kaufmann.

FOIL-D: Efficiently Scaling FOIL for Multi-relational Data Mining of Large Datasets

Joseph Bockhorst° and Irene Ong°

Department of Computer Sciences
University of Wisconsin, Madison WI 53706
joebock@cs.wisc.edu,ong@cs.wisc.edu

Abstract. Multi-relational rule mining is important for knowledge discovery in relational databases as it allows for discovery of patterns involving multiple relational tables. Inductive logic programming (ILP) techniques have had considerable success on a variety of multi-relational rule mining tasks, however, most ILP systems do not scale to very large datasets. In this paper we present two extensions to a popular ILP system, FOIL, that improve its scalability. (i) We show how to interface FOIL directly to a relational database management system. This enables FOIL to run on data sets that previously had been out of its scope. (ii) We describe estimation methods, based on histograms, that significantly decrease the computational cost of learning a set of rules. We present experimental results that indicate that on a set of standard ILP datasets, the rule sets learned using our extensions are equivalent to those learned with standard FOIL but at considerably less cost.

1 Introduction

Traditional data mining techniques aim to extract patterns from data sets that may be naturally represented in flat files. Conversely, relational databases represent data as a set of interconnected relational tables. Many useful patterns involve multiple tables, and since data in this format often cannot naturally be represented in flat files, traditional propositional data mining methods have difficulties learning such patterns.

Inductive Logic Programming (ILP) [1] algorithms aim to learn a set of first-order logical rules from multi-relational data and thus are well suited to multi-relational data mining tasks. There are however, two practical barriers that must be overcome before ILP systems may be applied to mining of large datasets. (i) ILP systems must deal with a limited amount of physical memory. Most ILP implementations, such as FOIL [2], tacitly assume the whole database fits into main memory. If it does not fit, these programs either crash or grind to an effective halt as they rely on the operating system to manage moving data between physical memory and disk. (ii) ILP systems must be faster. The

[0] Both authors contributed equally to this work

search space of ILP systems is very large and even heuristic methods are slow. Moreover, since the time needed to score an operator typically depends on the database size, direct application of most ILP systems to very large datasets is impractical.

In this paper, we describe extensions to the ILP algorithm FOIL that address both of these barriers. To deal with limited physical memory we leverage off the considerable effort that has been addressed to this issue in the design of relational database management systems (RDBMSs). We show how to succinctly express FOIL's operations in terms of SQL statements that RDBMSs have been optimized to execute. To deal with time cost we describe probabilistic models that we use to estimate the gain of FOIL's operators. We show how to use these models to significantly speed up learning. We present experimental results that indicate that with our estimation method we are able to learn the same rules as standard FOIL on several standard ILP datasets, but in significantly less time.

The interest in scaling up ILP for relational data mining in large datasets has been growing as Dimaio and Shavlik [3], Tang et. al [4] and Mooney et. al [5] have recently published research in this direction. Although the idea of incorporating the ability to learn first order rules from RDBMSs is not new – Stonebraker et. al [6] added this feature to Postgres and Brockhausen and Morik [7] have directly implemented an ILP algorithm (RDT) to a RDBMS – the use of probabilistic models to estimate scores for learning rules has, to the best of our knowledge, not been done before. Similar estimation ideas have been used in the area of query-optimization in databases [8, 9].

2 FOIL

Quinlan's first-order inductive learner [2, 10] (FOIL) is a popular ILP algorithm that learns function-free first order rules for a target relation. FOIL requires as input a set of *extensionally* defined relations. That is, each input relation is defined by a listing of its tuples rather than *intensionally* as set of logical rules. This parallels closely the organization of data in relational databases where we can think of the tuples of a table as defining a relation. One of the input relations, which we refer to as POS, is designated as the target relation. Another input relation, which we refer to as NEG, is of the same arity as POS and contains tuples that do not belong to the target relation. These two relations define FOIL's *initial* positive and negative tuple set. The other relations $B1, ..., BN$ serve as background knowledge.

Given its input, FOIL learns rules of the form

$$POS(X_1, ..., X_m) \leftarrow L_1 \wedge L_2 \wedge ...$$

where each L_i is a (possibly negated) literal and $X_1 \ldots X_m$ are distinct variables. The goal of FOIL is to discover a set of rules that entails all of the tuples in POS and none of the tuples in NEG.

Table 1. The FOIL algorithm. Given a set of tuples, POS, in the target relation, a set of tuples, NEG, not in the target relation and sets for tuples $B_1, ..., B_M$ that define M background relations, FOIL returns a set of first-order rules for the target relation. The symbol $\Leftarrow$ indicates variable assignment and the symbol $\leftarrow$ indicates logical implication.

```
1: procedure FOIL(POS, NEG, B_1, ···, B_N )
2:   learnedRules ⇐ {}
3:   POS_unc ⇐ POS                          ▷ start with all positive tuples uncovered
4:   repeat                                                    ▷ begin FOIL's outer loop
5:     newRule ⇐ LEARNONERULE()
6:     update POS_unc                ▷ remove tuples in POS_unc covered by newRule
7:     add newRule to learnedRules
8:   until |POS_unc| == 0
9:   return learnedRules
10: end procedure

11: procedure LEARNONERULE()
12:   newRuleBody ⇐ {}                          ▷ start with general rule POS ← true
13:   POS_curr ⇐ POS_unc                ▷ all positive tuples are covered by newrule
14:   NEG_curr ⇐ NEG                    ▷ all negative tuples are covered by newrule
15:   repeat                                                    ▷ begin FOIL's inner loop
16:     bestLit ⇐ getBestLiteral(candidateLiterals())
17:     conjoin bestLit to newRuleBody                ▷ add literal to growing rule
18:     update POS_curr                          ▷ arity of POS_curr may increase
19:     update NEG_curr                          ▷ arity of NEG_curr may increase
20:   until |NEG_curr| == 0
21:   return POS ← newRuleBody
22: end procedure

23: procedure GETBESTLITERAL(C)                    ▷ C is set of candidate literals
24:   maxGain ⇐ −∞
25:   for all candLit ∈ C do
26:     p′ ⇐ |POS_curr| if candLit is added to newrule
27:     n′ ⇐ |NEG_curr| if candLit is added to newrule
28:     p++ ⇐number of tuples in POS_curr covered by candLit
29:     gain ⇐ GAIN(p′, n′, p++)
30:     if gain > maxGain then
31:       bestLiteral ⇐ literal
32:       maxGain ⇐ gain
33:     end if
34:   end for
35:   return bestLiteral
36: end procedure
```

Table 1 presents a high level outline of FOIL. FOIL is a covering algorithm that on each iteration of its outer loop, which starts at Line 4[1], adds a single clause to its learned rule set that logically entails (covers) some of the previously uncovered tuples in POS and none of the tuples in NEG. FOIL's method of building a single rule, shown in the LEARNONERULE procedure of Table 1, begins with the general rule $POS \leftarrow \mathsf{true}$, which covers all positive and negative tuples. This rule is then specialized by greedily conjoining literals one at a time to the body until the new rule covers no negative tuples.

One trait of FOIL that distinguishes it from most propositional supervised learning algorithms is the dynamic nature of the positive and negative example training sets. When deciding which literal to append to the body of the new rule, FOIL considers only the *current* positive and negative tuple sets, POS_{curr} and NEG_{curr} in Table 1. From one iteration to the next, these sets may shrink, grow, increase in arity or some combination thereof.

At the start of the LEARNONERULE procedure POS_{curr} is initialized to the tuples of POS that are not covered by any rule learned so far, POS_{unc}, and NEG_{curr} is set to all the tuples in NEG. Following the addition of the chosen literal (Line 17), FOIL updates POS_{curr} and NEG_{curr}. Let $L(N_1, ..., N_a, O_1, ..., O_b)$ be the chosen literal where N_i is a *new* variable that does not appear anywhere else in the new rule and O_i is an *old* variable that appears either in the head or a previously introduced literal. If L is unnegated, the arity of tuples in the updated POS_{curr} and NEG_{curr} sets increases by a, the number of new variables. A tuple t in POS_{curr} (or NEG_{curr}) gives rise to a (possibly expanded) tuple in the updated set for each tuple in the relation associated with the chosen literal that matches t on the arguments indicated by the old variables.

FOIL uses an information theoretic heuristic to determine which literal to append to the body of a growing rule. On each iteration of its inner loop, which starts on Line 15, FOIL chooses the literal that has maximum gain where it defines the gain of literal L_i as

$$gain(L_i) = p^{++} \times \left\{ \log(\frac{p'}{p'+n'}) - \log(\frac{p}{p+n}) \right\}.$$

Here p and n are the sizes of POS_{curr} and NEG_{curr}, p' and n' are the sizes of the updated POS_{curr} and NEG_{curr} if L_i were added and p^{++} is the number of tuples in POS_{curr} that are covered by L_i.

The main computational cost of FOIL comes from the evaluation of all the candidate literals every time a new literal is added to a growing clause. If the input data set cannot fit in main memory, management of the current positive and negative tuple becomes more complicated. In the next section we show how, if our data is stored in relational database tables, these tasks can be concisely expressed by a small number of SQL statements.

[1] This and all subsequent references to line numbers refer those in Table 1.

3 FOIL-D

In this section we present FOIL-D, our implementation of FOIL that interfaces directly with relational databases. FOIL-D assumes operations that involve manipulation of the relational data may not fit in main memory and thus provides database operations, in terms of SQL statements, for them. FOIL-D does however, assume that other operations, such as generating and storing the candidate literals, fit in main memory.

One difference between FOIL and our current implementation of FOIL-D is that for simplicity FOIL-D only considers unnegated literals while FOIL considers both unnegated and negated literals. There is nothing fundamental that prevents FOIL-D from considering negated literals though, and we plan to add support for them in the future.

3.1 Database organization and operations

Let the tuples defining the input relations be in database tables named POS, NEG, B1, ..., BN where the mapping between tables and the relations in Table 1 is the obvious one. If the data is not in such a format, temporary tables can be constructed. We store the uncovered positive tuples and the current positive and negative examples in database tables named POS_UNC, POS_CURR and NEG_CURR respectively. Due to the dynamics of the training sets, the number of columns of POS_CURR and NEG_CURR may change during the course of the algorithm. At any time though there is a one-to-one correspondence between the columns of POS_CURR (and NEG_CURR) and the distinct variables that have been introduced by either the head or a literal in the body of the growing clause.

The left-hand column of Table 2 lists the line numbers from Table 1 where FOIL-D issues database queries. The right-hand column gives the SQL statements[2] for the corresponding line. We now discuss each of these operations in turn.

Line 3 Here we initialize POS_UNC to all the tuples in POS. SQL statements of the from `CREATE` *new-table-name* `LIKE` *existing-table-name* create a new empty table that has the same column names and types as the existing table indicated.

Line 6 Here we remove from POS_UNC those tuples covered by the rule just learned. We first save the tuples of POS_UNC into the temporary table OLD_POS_UNC and recreate an empty POS_UNC. The `INSERT INTO` statement fills POS_UNC with those tuples of OLD_POS_UNC that do not have any matching tuple in POS_CURR, the expanded tuples covered by the new rule. The semantics of `LEFT JOIN` is to include at least one tuple from the "left" table (OLD_POS_UNC in this case) even if none of them are selected

[2] These statements are compatible with version 4.1 of MySQL (http://www.mysql.com)

Table 2. SQL statements used by FOIL-D. The left-hand-column indicates line numbers from Table 1 where FOIL-D issues database queries. The right-hand-column lists the corresponding SQL statements.

Line 3	`CREATE TABLE` POS_UNC `LIKE` POS
	`INSERT INTO` POS_UNC `SELECT` * `FROM` POS
Line 6	`ALTER TABLE` POS_UNC `RENAME` OLD_POS_UNC
	`CREATE TABLE` POS_UNC `LIKE` POS
	`INSERT INTO` POS_UNC `SELECT` *cols* `FROM` OLD_POS_UNC
	`LEFT JOIN` POS_CURR `ON` *cond* `WHERE` *cond′*
	`DROP TABLE` OLD_POS_UNC
Line 13	`CREATE TABLE` POS_CURR `LIKE` POS
	`INSERT INTO` POS_CURR `SELECT` * `FROM` POS_UNC
Line 14	`CREATE TABLE` NEG_CURR `LIKE` NEG
	`INSERT INTO` NEG_CURR `SELECT` * `FROM` NEG
Line 18	`ALTER TABLE` POS_CURR `RENAME` OLD_POS_CURR
	`CREATE TABLE` POS_CURR (...)
	`INSERT INTO` POS_CURR `SELECT` *cols* `FROM` OLD_POS_CURR, $rel(bestLit)$ `WHERE` *cond*
	`DROP TABLE` OLD_POS_CURR
Line 19	`ALTER TABLE` NEG_CURR `RENAME` OLD_NEG_CURR
	`CREATE TABLE` NEG_CURR (...)
	`INSERT INTO` NEG_CURR `SELECT` *cols* `FROM` OLD_NEG_CURR, $rel(bestLit)$ `WHERE` *cond*
	`DROP TABLE` OLD_NEG_CURR
Line 26	`SELECT COUNT` (*) `FROM` POS_CURR, $rel(candLit)$ `WHERE` *cond*
Line 27	`SELECT COUNT` (*) `FROM` NEG_CURR, $rel(candLit)$ `WHERE` *cond*
Line 28	`SELECT DISTINCT COUNT` (*) `FROM` POS_CURR, $rel(candLit)$ `WHERE` *cond*

with the condition in the ON clause, provided they pass the condition in the WHERE clause. These tuples have null values for any columns that come from the "right" table (POS_CURR). The statement FOIL-D issues selects only those tuples of OLD_POS_UNC that do not match any in POS_CURR by setting $cond'$ to col = null where col is a column in POS_CURR. The condition *cond* in the ON clause is a set of equality constraints between columns in OLD_POS_UNC and columns in POS_CURR that correspond to variables in the target relation.

Lines 13 and 14 Here we initialize POS_CURR and NEG_CURR, the current positive and negative example set.

Lines 18 and 19 Here we update POS_CURR and NEG_CURR after we add the highest scoring literal *bestLit* to the growing rule. The statements to update POS_CURR are analogous to those to update NEG_CURR. For simplicity, we only describe those for updating POS_CURR. Before getting the tuples with the INSERT INTO statement, we save the old tuples in a temporary table OLD_POS_CURR and create a new POS_CURR table that will hold the updated examples. The new POS_CURR table will have a column for each column in OLD_POS_CURR and each new variable in *bestLit*. The INSERT INTO statement joins OLD_POS_CURR with the relation of $bestLit, rel(bestLit)$. The condition *cond* in the WHERE clause is a conjunction of equalities, one for each old variable in *bestLit*, between columns in OLD_POS_CURR and the corresponding columns in $rel(bestLit)$. The projection *cols* lists the columns in OLD_POS_CURR along with the columns in $rel(bestLit)$ that correspond to the new variables in *bestLit*.

Lines 26-27 Here we compute counts p' and n' that, along with p^{++}, we need to compute the gain of the candidate literal *candLit*. The conditions *cond* in the WHERE clauses are conjunctions of equalities, one for each new variable in *candLit*, and are the same for both statements. If *candLit* becomes *bestLit*, *cond* will also be used in the INSERT INTO SELECT statements issued when updating POS_CURR and NEG_CURR on Lines 18 and 19.

Line 28 Here we compute p^{++}. The DISTINCT keyword assures that tuples in POS_CURR are counted at most once. The condition *cond* is the same as the one used on Line 26 to compute p.

3.2 Computational cost of FOIL-D

The primary computational cost of running FOIL-D on large databases comes from the database *join* operations used to execute the six conditional SELECT statements (Lines 6, 18, 19, 26, 27 and 28). A join operation ($r \bowtie s$) between tables r and s selects a subset of the tuples in the cross product $r \times s$ that match a specified set of constraints. The implementation of the join operation is a heavily studied topic in database research. See, for example, Ramakrishnan's textbook [11]. The cost of a join depends on a number of properties of the join such as the number and types of constraints (*eg*, $>, <, =$), the presence of any database indexes on the join columns, and the number of tables. In FOIL-D all joins involve only equality constraints (*equi-joins*) between two tables. FOIL-D does

perform joins with $k > 1$ equality constraints (k-column joins). In this paper, we are agnostic about the implementation of join but measure the computational cost by the total number of joins performed to learn a theory.

Inspection of Tables 1 and 2 reveals that the number of joins needed to learn a rule set of R rules with L total literals where C total candidates are considered is

$$\#\text{ of join operations} = R + 2L + 3C$$

The number of rules in a learned theory R is typically small, often less than five, and the number of literals L is also manageable, in the ten's at most. The number of candidates C, however, is often much larger and thus C dominates the others. The number of candidate literals considered at *each* step depends most strongly on the arity of the maximum arity relation and the number of old variables introduced so far [12]. For example, the number of candidate literals considered to append to a rule with 5 old variables and max arity of 3 is 136 [12]. FOIL-D uses type constraints on the arguments of the relations (or the columns in the database) which reduces the total number of candidate literals somewhat but not so much that it does not dominate in the above expression. Next, we describe extensions to FOIL-D that reduce the total number of join operations needed to learn a theory.

4 FOIL-DH

To get the counts p', n' and p^{++} needed to compute the gain of a candidate literal we only need the *number* of tuples in the result sets of the SQL statements listed in Table 2 that we execute on Lines 26-28. Thus, one way to reduce the total number of joins is to compute p', n' and p^{++} by more efficient means. The approach we consider here is, using histograms, to construct probabilistic models of the tuples in each table and to use these models to estimate the counts. We call this system "FOIL-D with histograms" or simply FOIL-DH.

We maintain a histogram for each column of POS_UNC, NEG, B1, ..., BN, POS_CURR and NEG_CURR. Let $h^{r.c}$ be the histogram for the column $r.c$ of table r. The domain of $h^{r.c}$ is the same as the domain of the type of column $r.c$. The count $h^{r.c}(v)$ for value v is the number of tuples in r for which the value of column $r.c = v$.

4.1 Estimating p' and n'

Now we show how, given the column histograms and an independence assumption, to quickly estimate the size of any k-column equi-join, such as the ones to get p' and n'.

The probability $p^{r.c}(v)$ that a randomly selected tuple in table r has value v in column c is

$$p^{r.c}(v) = h^{r.c}(v)/|r|.$$

For an equi-join between columns c of table r and c' of table s the probability that a randomly selected tuple from the cross product $r \times s$ satisfies the equality constraint, and thus is included in the result set, is

$$p(r.c, s.c') = \sum_v p^{r.c}(v) \times p^{s.c'}(v)$$

where the sum is over all values in the type of column $r.c$ (and $s.c$). The number of tuples in the result set is exactly given by

$$size(r \bowtie_1 s) = |r||s| \times p(r.c, s.c')$$

where $\bowtie_1$ indicates a 1-column join between r and s.

This is an exact calculation because the nature of the cross product guarantees that for a randomly selected tuple in $r \times s$, the value in a column from r is statistically independent of the value of a column from s. For multi-column joins the summary statistics in the histograms are not sufficient to exactly compute the size of the result set. If we assume, however, that the values of all columns in the join from the *same* table are statistically independent, we can estimate the size of a $k-$column join as

$$\hat{size}(r \bowtie_k s) = |r||s| \prod_{i=1}^{k} p(r.i, s.i) \tag{1}$$

where here without loss of generality we assume column $r.i$ is joined with $s.i$ for $1 \leq i \leq k$.

Our estimates of p' and n' for candidate literal *candLit* then are

$$\hat{p}' = \hat{size}(\text{POS\_CURR} \bowtie_b rel(candLit))$$

and

$$\hat{n}' = \hat{size}(\text{NEG\_CURR} \bowtie_b rel(candLit))$$

where here $rel(candLit)$ is the relation of *candLit* and b is the number of old variables in *candLit*.

4.2 Estimating p^{++}

In order to compute the gain of *candLit*, in addition to p' and n', we need p^{++}, the number of tuples in POS_CURR (pre-update) still covered by the new rule if we accept *candLit*. Thus, we need to estimate the number of tuples in $r =$ POS_CURR that give rise to at least one tuple in $r \bowtie_k s$ where s is $rel(candLit)$.

To estimate p^{++}, we first compute q, the probability a randomly selected tuple in $r \times s$ matches on join columns 2 through k given the independence assumptions as

$$q = \prod_{i=2}^{k} p(r.i, s.i)$$

where again we assume column $r.i$ is joined with $s.i$ for $1 \leq i \leq k$. If $k = 1$ we set q to 1.0. If we randomly pick with replacement j tuples from $r \times s$, the probability that at least one matches on join columns 2 through k is

$$m(j) = (1.0 - (1.0 - q)^j).$$

A tuple in r with value v in the first join column has $h^{s.1}(v)$ tuples in the cross product that match in the first column and this many "chances" to match on the remaining $k - 1$ columns. So, we estimate the probability that the tuple will have at least one match in the result set as $m(h^{s.1}(v))$. Summing over all values and multiplying by $h^{r.1}(v)$ gives our estimation of p^{++}:

$$\hat{p}^{++} = \sum_v h^{r.1}(v) * m(h^{s.1}(v)).$$

Our choice of doing this final sum over values of the first join column is arbitrary. We could have chosen any i of the k join columns as the one to do the final sum over where then we calculate q as the probability of a match on the $k - 1$ columns excluding join column i. Due to errors introduced by the sampling with replacement assumption of $m(j)$, in general the estimate $\hat{p}^{++}$ will be different for each choice of final join column i. In practice however, we have found $\hat{p}^{++}$ to be close for any choice of i.

As with the expression for estimating p' and n', our estimation $\hat{p}^{++}$ is exact for 1-column joins. Thus, we can exactly compute the gain for candidate literals with exactly 1 old variable given the column histograms.

4.3 Estimating the highest gain literal

We speed up FOIL-D by using estimations of p', n' and p^{++} to estimate the highest scoring candidate literal in two ways. The first method simply estimates the gain of every candidate literal directly with $\hat{p}', \hat{n}'$ and $\hat{p}^{++}$ and chooses the one with highest estimated gain to append to the growing clause. This method eliminates the $3C$ joins needed to estimate the gain of the C candidate literals thereby reducing the number of joins performed to learn a theory with R rules and L literals to

$$\#\text{ of join operations} = R + 2L.$$

A problem with this method is that the independence assumption often causes the estimated gains of the highest-scoring candidate literals with 2 or more old variables to be less than their true gains. In cases where the candidate literal with highest gain has multiple old variables, this procedure often erroneously estimates the highest scoring literal to be one with only 1 old variable.

We find, however, the relative ranks of the candidate literals with multiple old variables, especially the highest scoring ones, is relatively stable following the estimation procedure. This leads to our other use of $\hat{p}', \hat{n}'$ and $\hat{p}^{++}$.

We also use the estimates of the candidate literal gains as a filter to reduce the number of candidate literals whose gains are computed exactly. Given filter size

F we pick a candidate literal to conjoin to the growing clause with this method as follows. First, we compute the estimated gains of the entire candidate set using our histograms as described above. Next, we rank the candidate literals with two or more old variables by estimated gain and compute the exact gain for the top F. Finally, we choose that the literal with the highest true gain among these F and the top scoring candidate literal with a single old variable. We call this method FOIL-DH(F).

The number of joins performed to learn a theory with FOIL-DH(F) is

$$\#\text{ of join operations} = R + 2L + 3FL$$

which is still much smaller than $R + 2L + 3C$ for small F.

4.4 Richer Probabilistic Models

As just described, the current implementation of FOIL-DH estimates the size of a multi-column join by assuming that, for a randomly selected tuple from either of the tables in the join, the values for the join columns are independent of one another. If the join columns are highly correlated, this assumption is likely to lead to inaccurate gain estimates and could result in poor rules.

One way to address this pitfall is to explicitly represent the dependencies between columns with a more complex probability model, such as a *Bayesian network*[3][13], for each table's tuples. Using Bayesian networks, the product on right-hand-side of the Equation 1 would be replaced with the probability, as given by the Bayesian networks for the two tables, that a randomly selected tuple from the cross-product would match on the join columns.

5 Experimental results

This section describes experiments conducted and results obtained by FOIL-D, FOIL-DH and FOIL-DH(F) on three learning tasks taken from machine learning literature, which were used by [2] to illustrate the power of FOIL. Descriptions of the domains are taken from [2]. The aim of the experiments performed is to determine whether the cost of obtaining accurate hypotheses, with respect to FOIL, can be significantly reduced by using probabilistic models (FOIL-DH) and filtering (FOIL-DH(F)) to estimate the scores of literals.

5.1 Learning connectivity of a network

Our first learning task involved learning the definition of *can-reach* in the network from [2], shown in Figure 1. Extensional definitions of *can-reach*(N,N), *not-can-reach*(N,N) as well as *linked-to*(N,N) were given as positive example,

[3] The current version of FOIL-DH corresponds to a Bayesian network for each table that has one vertex for each column and no edges.

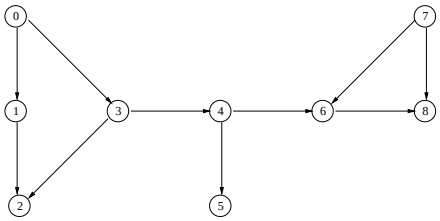


Fig. 1. A small network, where ⟵ indicates *linked-to*, from [2]

Table 3. Rules learned for *can-reach*

FOIL-D, FOIL-DH(1-6)	can-reach(X0, X1) ← linked-to(X0, X1) can-reach(X0, X1) ← linked-to(X0, X2), can-reach(X2, X1)
FOIL-DH(0)	can-reach(X0, X1) ← linked-to(X0, X2), linked-to(X3, X1), linked-to(X2, X4), can-reach(X2, X1) can-reach(X0, X1) ← linked-to(X0, X2), linked-to(X3, X1), linked-to(X2, X4), linked-to(X4, X5), linked-to(X6, X3)

negative example and background knowledge respectively. Variables in this relation consists of one type N (Node). The goal is to learn a general definition for *can-reach*. Table 3 shows the rules learned by FOIL-D and FOIL-DH(0-6), which are using filter sizes set to 0 through 6.

5.2 Learning eastbound trains

The second learning task is from the INDUCE system by [14] and described by [2]. Trains in Figure 2 have different numbers of cars, with various shapes and tops, carrying loads of various number and shape. The task is to distinguish between *eastbound* and *westbound* trains. The target relation *eastbound*(T) is to be defined in terms of the following relations has_car(T,C), not_has_car(T,C), in_front(C,C), behind(C,C), long(C), short(C), open_rectangle(C), not_open_rectangle(C), hexagon(C), not_hexagon(C), bucket(C), not_bucket(C), ellipse(C), not_ellipse(C) rectangle(C), not_rectangle(C), u_shaped(C), not_u_shaped(C), jagged_top(C), not_jagged_top(C), peaked_top(C), not_peaked_top(C), flat_top(C), not_flat_top(C), arc_top(C), not_arc_top(C), open_top(C), closed_top(C), contains_no_load(C), contains_load(C,LS), one_load(C), two_load(C), three_load(C), two_wheels(C), three_wheels(C), double(C), and not_double(C).

The variables are of three types: T (train_type), C (car_type) and LS (load_shape_type). Clauses learned by FOIL-D and FOIL-DH(0-6) are shown in Table 4.

5.3 Learning family relationships

The third and final task we consider is that of learning family relationships, described by [15]. Figure 3 shows two isomorphic families of twelve members each.

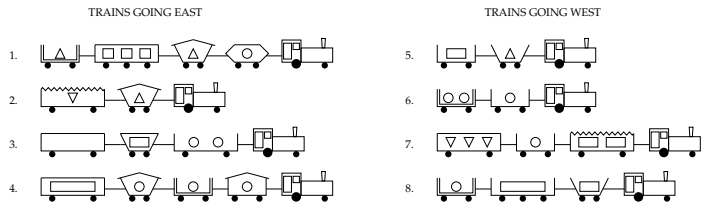


Fig. 2. Eastbound and westbound trains from [14] and [2]

Table 4. Rules learned for *eastbound*

FOIL-D, FOIL-DH(0-6)	eastbound(X0) ← has-car(X0, X1), closed-top(X1), short(X1)

There are twelve relationship types to be learned: *mother*, *father*, *wife*, *husband*, *son*, *daughter*, *sister*, *brother*, *aunt*, *uncle*, *niece* and *nephew*. Each target relation to be learned are to be defined in terms of the following background relations: mother(P,P), father(P,P), wife(P,P), husband(P,P), son(P,P), daughter(P,P), sister(P,P), brother(P,P), aunt(P,P), uncle(P,P), niece(P,P), nephew(P,P), and their negations. The variables are all of type P (People).

The rules learned by FOIL-DH(1) for this task are identical to those learned by FOIL-D on all twelve relations. Table 5 and Table 6 show the definitions learned by FOIL-D and FOIL-DH(0-6) for *uncle* and *mother* respectively. Since in these families, all uncles are married, the second literal in both *uncle* clauses serves the purpose of asserting that person X0 is a man. The rules learned by FOIL-D and FOIL-DH(1-6) for other relations are similar in structure. FOIL-DH(0) fails to learn any rules[4]

Table 5. Rules learned for *uncle*

FOIL-D, FOIL-DH(1-6)	uncle(X0, X1) ← niece(X1, X0), husband(X0, X2)
	uncle(X0, X1) ← nephew(X1, X0), husband(X0, X2)
FOIL-DH(0)	No rules learned

When learning the relations *can-reach* and the twelve familial relationships, FOIL-D as well as FOIL-DH(1-6) constructs accurate, compact rules. However, for the *can-reach* relation, FOIL-DH(0) generates long, convoluted clauses that consist of only literals with *one* old variable. Furthermore, FOIL-DH(0) does not

[4] FOIL-DH(0) learns no rules in cases where the first rule it "learns" covers zero positive examples and is discarded.

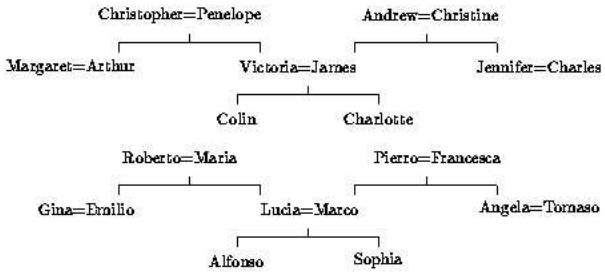


Fig. 3. Two family trees, where = means *married-to*, from [15] and [2]

Table 6. Rules learned for *mother*

FOIL-D, FOIL-DH(1-6)	mother(X0, X1) ← daughter(X1, X0), husband(X2, X0) mother(X0, X1) ← son(X1, X0), husband(X2, X0)
FOIL-DH(0)	No rules learned

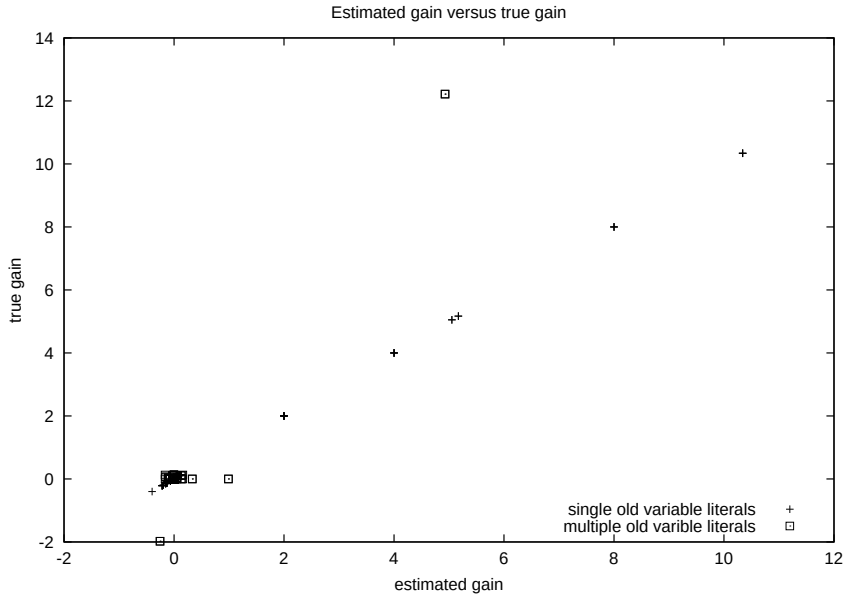


Fig. 4. Estimated gain versus actual gain calculations for adding the first literal in the relation *uncle*. The literal with the highest scoring true gain, *niece(X1,X0)*, has true gain of near 12 and estimated gain of near 5. This literal is not chosen by FOIL-DH(0), but since this literal has the highest estimated score of the multiple old variable literals it is correctly chosen by FOIL-DH(1).

generate any rules for any of the twelve family relationships. The reason for this can be seen in Figure 4. Whenever FOIL-DH(0) makes an estimation of the gain for literals with *multiple* old variables, it is always *under* estimates the actual gain of the highest scoring literals, whereas it always estimates the *exact* actual gain for literals with exactly *one* old variable. Hence, literals with exactly *one* old variable will score higher than literals with *multiple* old variables, resulting in the unique property of the rules learned by FOIL-DH(0) for the relation *can-reach*. This reasoning also supports the results of FOIL-DH(0) on the relationships of the family dataset because the definitions of those relations require literals with *two* old variables to be added first to the body of the clause.

How then was FOIL-DH(1), with a filter size of just 1, able to learn the correct definitions for *can-reach* and the twelve familial relationships? One possible explanation is that even though the estimations for literals with multiple old variables are lower, the order of the estimated gain for these literals is maintained. This would allow FOIL-DH(1), which specifically considers the highest estimated literal with *multiple* old variables, to accurately select the best literal.

Surprisingly, all the different FOIL types in our experiments were able to learn the *exact* rule for *eastbound* trains as shown in Table 4. It is interesting to note that FOIL-DH(0) was able to learn the exact rule for *eastbound* trains, in accordance with our reasoning above, because the rule is comprised of literals with exactly *one* old variable.

Figure 5 shows the total number of JOINs performed for the different FOIL types. FOIL-DH(0) to FOIL-DH(6) grows linearly in the number of JOINs performed and they still provide significantly more savings than FOIL-D.

6 Conclusion

We have presented preliminary methods towards enabling the ILP system FOIL to be applied to multi-relational data mining tasks on large data sets. Our methods address both the space and time hurdles that prohibit standard ILP implementations from being applied to these problems.

To deal with insufficient physical memory we build off of relational database management systems. We have described FOIL-D, a system that mimics the operation of FOIL but that performs the memory intensive FOIL operations using SQL queries to a relational database.

To deal with the slowness of FOIL on very large datasets we have described FOIL-DH, an extension of FOIL-D that uses histograms to quickly estimate the gains of candidate literals without performing expensive database join operations. We also showed how we use the estimation procedure as a filter to select a small number of candidate literals whose gain we compute exactly.

Our experimental results show that while FOIL-DH dramatically reduces the numbers of joins it sometimes fails to learn the correct theories on a set of standard ILP problems because of erroneous estimates. If, however, we use the estimates as a filter we are able to learn the same correct theories as FOIL-D on the datasets we looked at while performing significantly fewer joins.

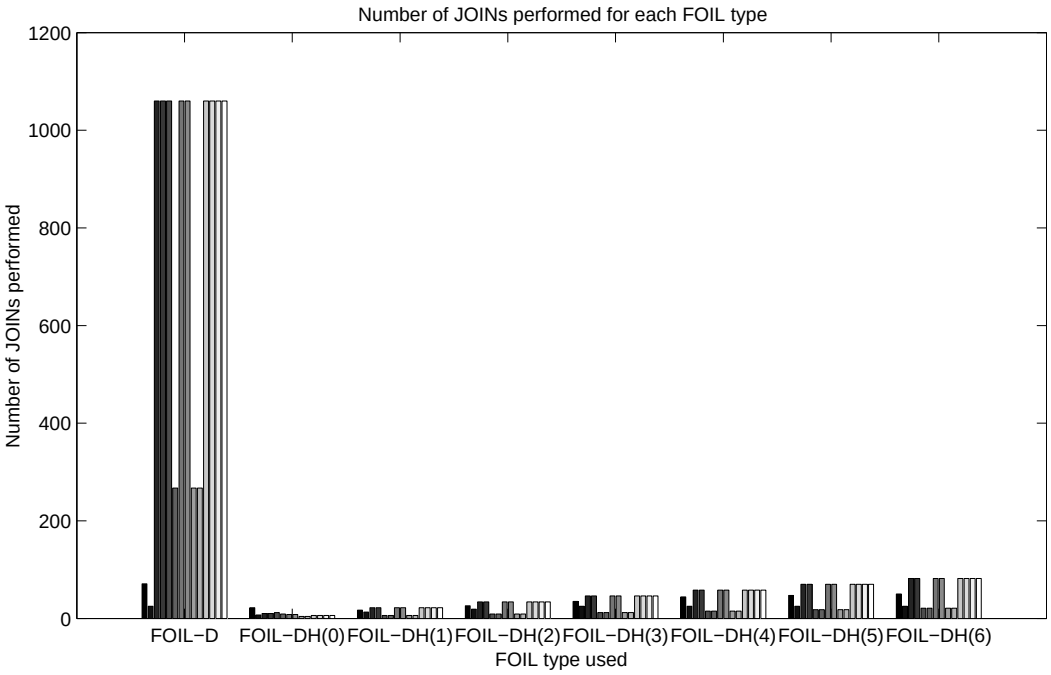


Fig. 5. Total number of JOINs performed for the different FOIL types when learning the following relations (in order from left to right in the histogram): can-reach, east-bound, mother, father, wife, husband, son, daughter, sister, brother, aunt, uncle, niece, nephew

The experiments we present in this paper all involve small datasets on standard but contrived problems. Our plans for the future involve evaluating FOIL-D and FOIL-DH in terms of accuracy and efficiency on a number of large real-world datasets. In addition, we plan on investigating more complex probability models, based on Bayesian networks.

In closing we note that although this paper has dealt exclusively with extending FOIL, some of the concepts developed here are applicable to other ILP systems. For example, systems that score whole clauses (as opposed to FOIL's scoring of literals) as a function of the number of positive and negative examples covered may benefit from similar estimation methods.

7 Acknowledgments

We gratefully acknowledge support for this research from U.S. Air Force grant F30602-01-2-0571 and NIH grant T15-LM07359-01. Thanks to David Page, Vítor Santos Costa and Inês Dutra who helped and encouraged us to present the ideas in this paper. Thanks also to Rich Maclin for help with graph formatting.

References

1. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994)
2. Quinlan, J.R.: Learning logical definitions from relations. Machine Learning **5** (1990) 239–2666
3. Dimaio, F., Shavlik, J.: Speeding up relational data mining by learning to estimate candidate hypothesis scores. In: Proceedings of the ICDM Workshop on Foundations and New Directions of Data Mining. (2003)
4. Tang, L.R., Mooney, R.J., Melville, P.: Scaling up ilp to large examples: Results on link discovery for counter-terrorism. In: Proceedings of the KDD-2003 Workshop on Multi-Relational Data Mining, Washington, DC (2003) 107–121
5. Mooney, R.J., Melville, P., Tang, L.R., Shavlik, J., Dutra, I., Page, D., Santos Costa, V.: Relational data mining with inductive logic programming for link discovery. In Kargupta, H., Joshi, A., Sivakumar, K., Yesha, Y., eds.: Data Mining: Next Generation Challenges and Future Directions. Volume To appear. AAAI Press (2004)
6. Stonebraker, M., Kemnitz, G.: The postgres next-generation database management system. Communications of the ACM **34** (1991) 78–92
7. Brockhausen, P., Morik, K.: Directaccess of an ilp algorithm to a database management system. In: Proceedings of the MLnet Familiarization Workshop. (1996) 95–110
8. Ioannidis, Y.E., Poosala, V.: Histogram-based solutions to diverse database estimation problems. IEEE Data Eng. Bull. **18** (1995) 10–18
9. Ioannidis, Y.E., Poosala, V.: Balancing histogram optimality and practicality for query result size estimation. In: SIGMOD Conference. (1995) 233–244
10. Quinlan, J.R., Cameron-Jones, R.M.: FOIL: A midterm report. In: Proceedings of the European Conference on Machine Learning, Vienna, Austria (1993) 3–20

11. Ramakrishnan, R.: Database Management Systems. McGraw-Hill, New York (1998)
12. Pazzani, M., Kibler, D.: The utility of knowledge in inductive learning. Machine Learning **9** (1992) 57–94
13. Pearl, J.: Probabalistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA (1988)
14. Michalski, R.S., Mozetič, I., Hong, J., Lavrač, N.: The multipurpose incremental learning system AQ15 and its testing application to three medical domains. In: Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA, Morgan Kaufmann (1986) 1041–1045
15. Hinton, G.E.: Learning distributed representations of concepts. In: Proceedings of the Eighth Annual Conference of the Fifth International Joint Conference on Artificial Intelligence, Amherst, MA, Lawrence Erlbaum (1986) 356–362

Using Bayesian Classifiers to Combine Rules

Jesse Davis, Vítor Santos Costa, Irene M. Ong,
David Page and Inês Dutra

Department of Biostatistics and Medical Informatics
University of Madison-Wisconsin
{jdavis, vitor, ong, page, dutra}@biostat.wisc.edu

Abstract. One of the most popular techniques for multi-relational data mining is Inductive Logic Programming (ILP). Given a set of positive and negative examples, an ILP system ideally finds a logical description of the underlying data model that discriminates the positive examples from the negative examples. However, in multi-relational data mining, one often has to deal with erroneous and missing information. ILP systems can still be useful by generating rules that captures the main relationships in the system. An important question is how to combine these rules to form an accurate classifier. An interesting approach to this problem is to use Bayes Net based classifiers. We compare Naïve Bayes, Tree Augmented Naïve Bayes (TAN) and the Sparse Candidate algorithm to a voting classifier. We also show that a full classifier can be implemented as a CLP($\mathcal{BN}$) program [14], giving some insight on how to pursue further improvements.

1 Introduction

The last few years have seen a surge of interest in multi-relational data mining, with applications in areas as diverse as bioinformatics and link discovery. One of the most popular techniques for multi-relational data mining is Inductive Logic Programming (ILP). Given a set of positive and negative examples, an ILP system ideally finds a logical description of the underlying data model that differentiates between the positive and negative examples. ILP systems confer the advantages of a solid mathematical foundation and the ability to generate understandable explanations.

As ILP systems are being applied to tasks of increasing difficulty, issues such as large search spaces and erroneous or missing data have become more relevant. Ultimately, ILP systems can only expect to search a relatively modest number of clauses, usually on the order of millions. Evaluating increasingly complex clauses may not be the solution. As clauses grow larger, they become more vulnerable to the following errors: a query will fail because of missing data, a query will encounter an erroneous database item, and a clause will give correct answers simply by chance.

Our work relates to a sizeable application in the field of link discovery. More precisely, our concern involves finding aliases in a relational domain [15] where

the data is subject to high levels of corruption. As a result, we cannot hope that the learned rules will generally model the entire dataset. In these cases, ILP can at best generate rules that describe fragments of the underlying model. Our hope is that such rules will allow us to observe the central relationships within the data.

An important question is how to combine the partial rules to obtain a useful classifier. We have two major constraints in our domain. First, we expect the number of positives to grow linearly with the number of individuals in the domain. In contrast, the number of negatives increases with the number of pairs, and therefore grows quadratically. Consequently, any approach should be robust to false positives. Furthermore, flexibility is also an important consideration as we ultimately want to be able to weigh precision versus recall through some measure of confidence, ideally in the form of a probability. Secondly, we expect to use the system for different datasets: our method should not be prone to overfitting and it should be easy to parameterize for datasets with different observabilities and error rates.

The previous discussion suggests probabilistic-based classifiers as a good approach to our problem. We explore three different Bayes net based approaches to this problem. Each ILP learned rule is represented as a random variable in the network. The simplicity and robustness of the Naïve Bayes classifier make it a good candidate for combining the learned rules [12]. Unfortunately, Naïve Bayes assumes independence between features and our rules may be quite interdependent and perhaps even share literals. A natural extension is to use TAN [6] classifiers as they offer an efficient way to capture dependencies between rules. Additionally, we explore using the Sparse Candidate algorithm [7] for learning the structure of a full Bayes net. An alternative approach we consider is to group our rules as an ensemble [3] and use voting, which has had excellent results in practice. We will evaluate the relative merits of these approaches.

The paper is organized as follows. We first discuss the problem in more detail. Then, we explain the voting and Bayesian based approaches to rule combination. Next, we present the main applications and discuss our results. We follow this by demonstrating how we can represent Bayesian classifiers as a logic program with probabilities, using CLP($\mathcal{BN}$). Finally, we end with related work and our conclusions.

2 Using ILP

From a logic perspective, the ILP problem can be defined as follows. Let E^+ be the set of positive examples, E^- be the set of negative examples, $E = E^+ \wedge E^-$, and B be the background knowledge. In general, B and E can be arbitrary logic programs. The aim of an ILP system is to find a set of hypotheses (also referred to as a theory) H, in the form of a logic program, such that all positive examples and none of the negative examples are covered by the program.

In practice, learning processes generate relatively simple clauses which only cover a limited subset of E^+. Moreover, such clauses often cover some examples

in E^-. One possible reason for the presence of these errors is that these examples may have been misclassified. A second reason is that approximated theories can never be as strict as the ground truth: if our clause is only a subclause of the actual explanation, it is possible that the clause will cover a few other incorrect examples. We also have to address implementational difficulties: for most cases we can only search effectively for relatively simple explanations (clauses). Therefore, we assume that clauses represent fragments of the ground-truth and that the learning process can capture different "features" of the ground truth. Clauses have some distribution, which is likely to be non-uniform, over the interesting aspects of the ground-truth theory. Even if we do not capture all features of the ground truth, we can still learn interesting and relevant clauses.

Given a set H of clauses learned in an incomplete world we can combine them to obtain a better classifier. One possible approach to combine clauses would be to assume that each clause is an explanation, and form a disjunction over the clauses. Although this approach has the merit of simplicity, and should work well for cases where we are close to the ground truth, it does have two serious issues we need to consider:

- We are interested in applications where the number of false instances dominates. Unfortunately, the disjunction of clauses maximizes the number of false positives.
- We expect the classifier to make mistakes, so ideally we would like to know the degree of confidence we have in a classification.

Our problem is not novel, and several approaches come to mind. We shall focus on two such approaches here. The idea of exploiting different aspects of an underlying classifier suggests ensemble-based techniques. Previous work on applying ensemble methods to ILP [4] suggests that exploring the variability in the seed is sufficient for generating diverse classifiers. We thus decided to use a simple approach where we use the ILP engine to generate clauses and then use voting to group them together. A second alternative is to consider each clause as a feature of an underlying classifier. We want to know which features are most important. Several possibilities exist and we focus on Bayesian networks, as they provide us with an estimated probability for each different outcome.

3 Combining Rules

3.1 Voting

It is well known that ILP systems that learn clauses using seeds are exploiting different areas of the search space, in a manner analogous to ensemble methods. In this vein, recent ILP work has exploited several techniques for ensemble generation, such as bagging or bootstrapping [4] and different forms of boosting [5,13,9,10]. Bagging is a popular ensemble method that consists of generating different training sets where each set contains a sample, with replacement, of the original dataset. Hypotheses are learned from each dataset, and combined

through a voting method. Alternatively, in boosting each classifier is built depending on the errors made by previous classifiers. Each new rule thus depends on the performance of the previous one.

Previous work on applying bagging to ILP [4] suggests that exploring variability from using different seed examples can be sufficient for generating diverse classifiers. We shall follow a similar approach: we use hypotheses generated from different runs of the ILP system, and combine them through unweighted voting. With this method, we consider an example to be positive depending on the number of clauses that are satisfied for that example. The number of clauses we need to satisfy to classify the example as positive is a variable threshold parameter. One major advantage of using a voting method is that we can obtain different values of precision and recall by varying the voting threshold. Thus, although a voting method does not give an estimate of the probability for each classification, it does provide an excellent baseline to compare with Bayesian-based methods.

3.2 Bayesian Networks

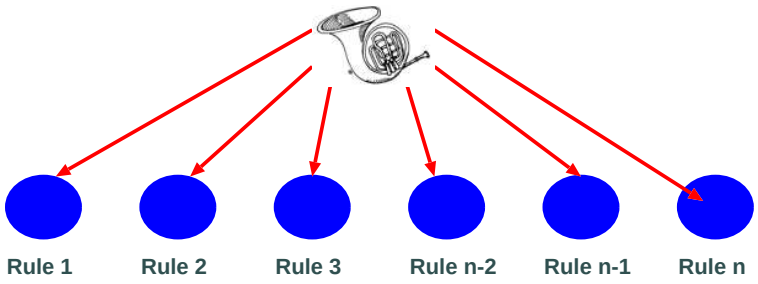


Fig. 1. A Naïve Bayes Net.

We expect every learned clause to be related to a clause in the "true" theory. Hence, we would also expect that the way each learned clause classifies an example is somehow dependent on the example's true classification. This suggests a simple approach where we represent the outcome for each clause as a random variable, whose value depends on the example's classification. The Naïve Bayes approach is shown in Figure 1 [12]. Advantages of this approach are that it is straightforward to understand as well as easy and fast to train.

The major drawback with Naïve Bayes is that it makes the assumption that the clauses are independent given the class value. Often, we expect clauses to be strongly related. Learning a full Bayes Net is an NP-complete problem, so in this work, we experimented with Tree Augmented Naïve Bayes (TAN) [6] networks. Figure 2 shows an example of a TAN network. TAN models allow for more complex network structures than Naïve Bayes. The model was proposed by Geiger in 1992 [8] and it extends work done by Chow and Liu [2]. Friedman,

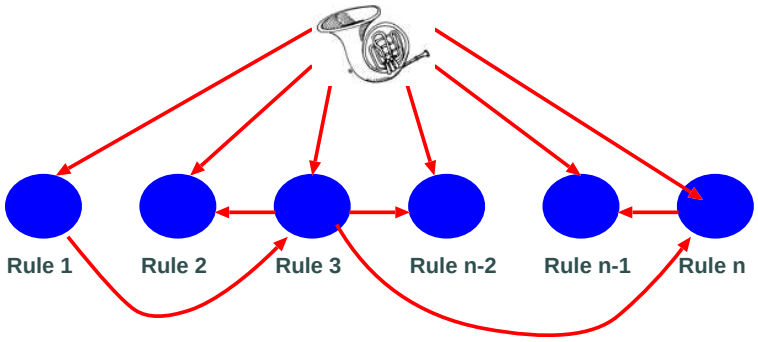


Fig. 2. A TAN Bayes Net.

Geiger and Goldszmidt [6] evaluated the algorithm on its viability for classification tasks. The TAN model, while retaining the basic structure of Naïve Bayes, also permits each attribute to have at most one other parent, allowing the model to capture dependencies between attributes. To decide which arcs to include in the 'augmented' network, the algorithm makes a complete graph between all the non-class attributes, where the weight of each edge is given as the conditional mutual information between those two attributes. A maximum weight spanning tree is constructed over this graph, and the edges that appear in the spanning tree are added to the network. Geiger proved that the TAN model can be constructed in polynomial time with a guarantee that the model maximizes the Log Likelihood of the network structure given the dataset.

The problem arises of whether different Bayes networks could do better. We report on some preliminary work using the Sparse Candidate Algorithm [7]. The Sparse Candidate algorithm tries to speed up learning a full Bayesian Network by limiting the search space of possible networks. The central premise is that time is wasted in the search process by evaluating edges between attributes that are not highly related. The algorithm retains standard search techniques, such as greedy hill climbing, but uses mutual information to limit the number of possible parents for each attribute to small 'candidate' set. The algorithm works in two phases. In the first phase, the candidate set of parents is picked for each attribute. The candidate set must include all current parents of a node. The second step involves performing the actual search. These two steps are repeated either for a set number of times or until the score of the network converges.

4 Results

This section presents our results and analysis of the performance of several applications. For each application we show precision versus recall curves for the four methods: Naïve Bayes, TAN, Sparse Candidate and voting. All our experiments were performed using Srinivasan's Aleph ILP system [16] running on the Yap

Prolog system. We used our own software for Naïve Bayes and TAN. For the Sparse Candidate Algorithm we used the LearnBayes program provided by Nir Friedman and Gal Elidan. For this algorithm we set the number of candidate parents to be five and we used the Bayesian Information Criterion as the scoring function. All results are obtained using five fold cross-validation.

Our main experiment was performed on synthetic datasets developed by Information Extraction & Transport, Inc. within the EAGLE Project [15,11]. The datasets are generated by simulating an artificial world with large numbers of relationships between agents. The data focuses on *individuals* which may have capabilities, belong to groups, and participate in a wide range of *events*. In our case, given that some individuals may be known through different identifiers (e.g., through two different phone numbers), we were interested in recognizing whether two identifiers refer to the same individual.

All datasets were generated by the same simulator, but with different parameters for observability (how much information is available as evidence), corruption, and clutter (irrelevant information that is similar to the information being sought). Five datasets were provided for training, and six for evaluation. All the datasets include detailed data on a few individuals, including aliases for some individuals. Depending on the dataset, the data may or may not have been corrupted.

Our methodology was as follows. First, we used the five training datasets to generate rules, using the ILP system Aleph. Using the rules learned from the training set, we selected the ones with best accuracy and combined them with domain expert knowledge to provide new feedback to the training phase. Using the final set of learned rules, we converted each of the evaluation datasets into a set of propositional feature vectors, such that each rule appeared as an attribute in the feature vector. Each rule served as a boolean attribute, which received a value of one if the rule matched the example and zero otherwise. For each of the six test datasets, we performed five fold cross validation. The network structure and parameters were learned on four of the folds, while the accuracy was tested on the remaining fold. For each dataset, we fixed the ratio of negative examples to positive examples at seventy to one. This is an arbitrary ratio since the full datasets are exceedingly large, and the ground truth files were only recently released.

The precision/recall (P/R) curves for the different datasets are seen in Figures 3 through 8. On each curve, we included 95% confidence intervals on the precision score for select levels of recall. The curves were obtained by averaging the precision and recall values for fixed thresholds. The precision recall curve for the TAN algorithm dominates the curves for Naïve Bayes and voting on all six of the datasets. For each dataset, there are several places where TAN yields at least a 20 percentage point increase in precision, for the same level of recall, over both Naïve Bayes and voting. On two of the six datasets, Naïve Bayes beats voting, while on the remaining four they have comparable performance. One reason for TAN's dominance compared to Naïve Bayes is the presence of rules which are simply refinements of other rules. The TAN model is able to capture some

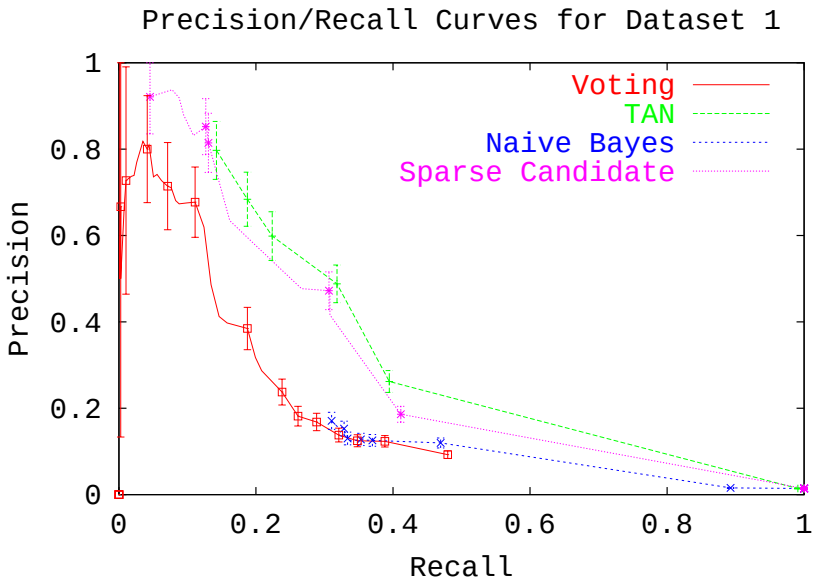


Fig. 3. P/R for Dataset 1

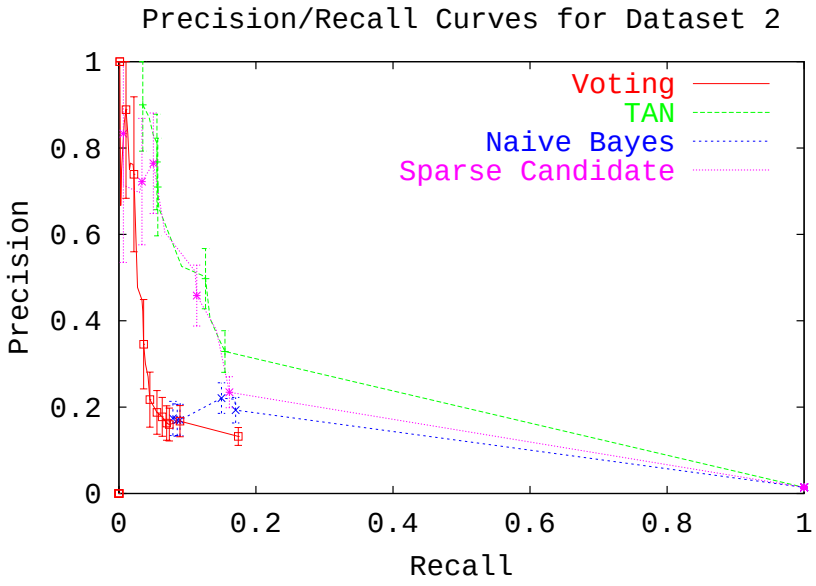


Fig. 4. P/R for Dataset 2

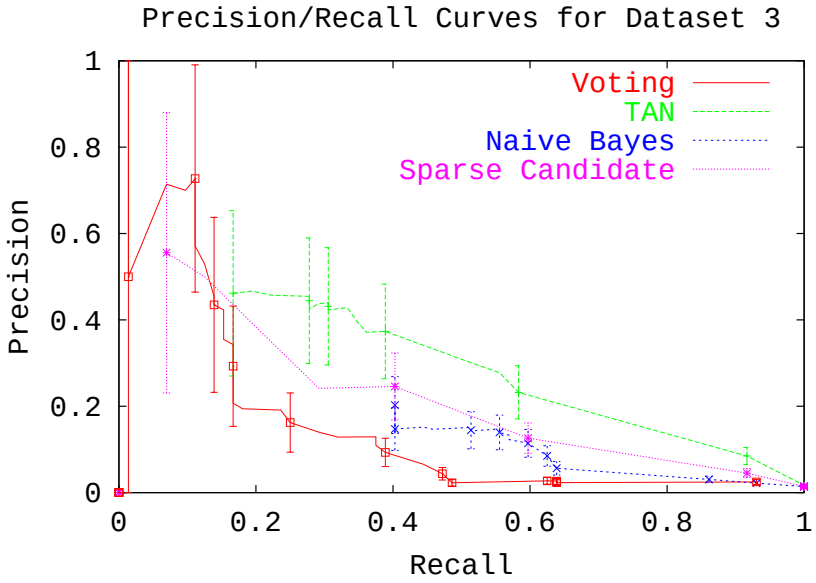


Fig. 5. P/R for Dataset 3

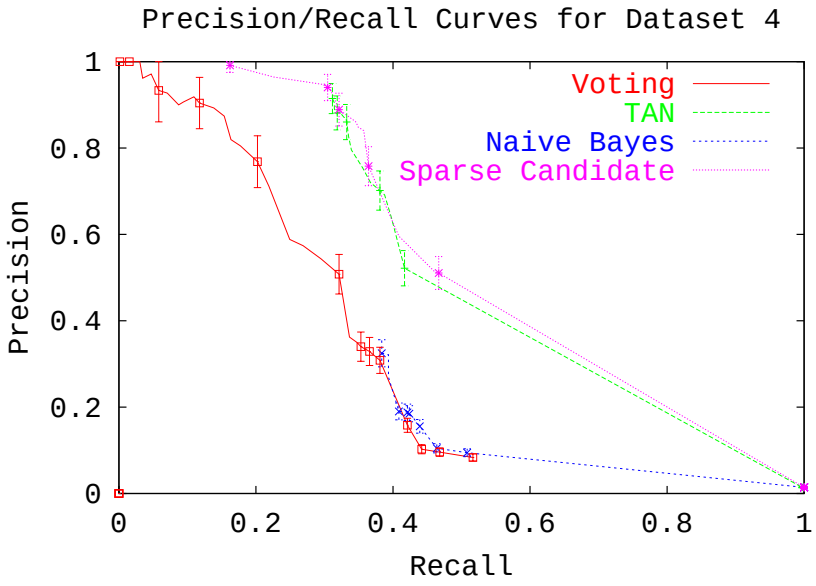


Fig. 6. P/R for Dataset 4

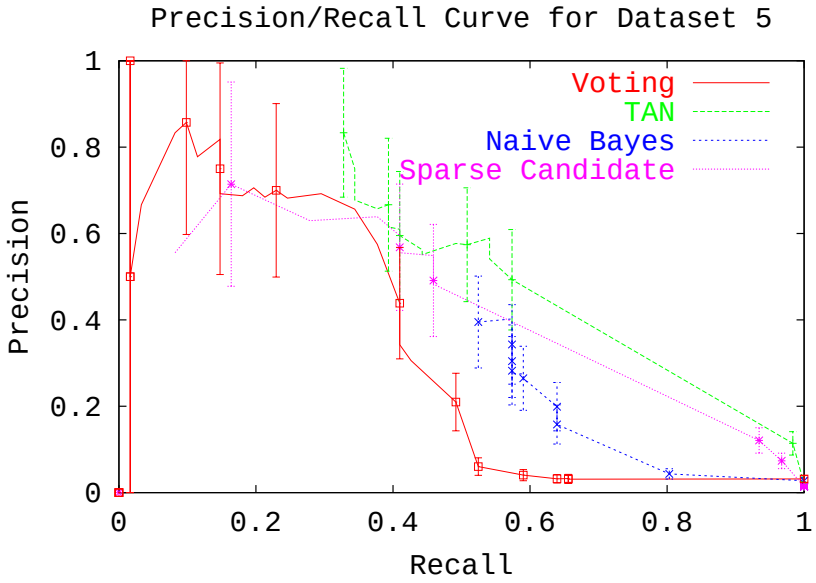


Fig. 7. P/R for Dataset 5

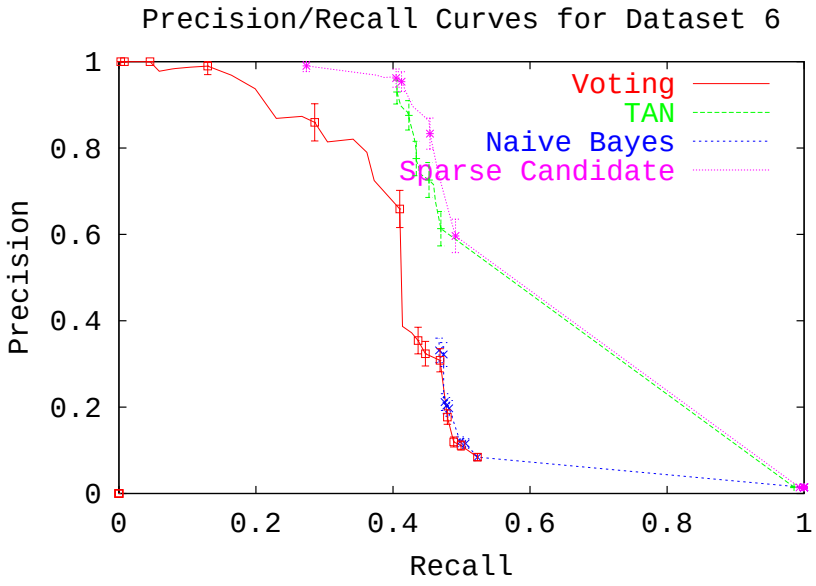


Fig. 8. P/R for Dataset 6

of these interdependencies, whereas Naïve Bayes explicitly assumes that these dependencies do not exist. Naïve Bayes' independence assumption accounts for the similar performance compared to voting on several of the datasets. TAN and the Sparse Candidate algorithm had similar precision recall curves. The package we used for the Sparse Candidate algorithm only allows for building generative models. TAN is a discriminative model, so it emphasizes differentiating between positive and negative examples. An important follow-up experiment would be to adapt the Sparse Candidate algorithm to use discriminative scoring functions.

In situations with imprecise rules and a preponderance of negative examples, such as these link discovery domains, Bayesian models and especially TAN provide an advantage. One area where both TAN and Naïve Bayes excel is in handling imprecise rules. The Bayes nets effectively weight the precision of each rule either individually or based on the outcome of another rule in the case of TAN. The Bayesian nets further combine these probabilities to make a prediction of the final classification, allowing them to discount the influence of spurious rules in the classification process. Ensemble voting does not have this flexibility and consequently lacks robustness to imprecise rules. Another area where TAN provides an advantage is when multiple imprecise rules provide significant overlapping coverage on positive examples and a low level of overlapping coverage on negative examples. The TAN network can model this scenario and weed out the false positives. One potential disadvantage to the Bayesian approach is that it could be overly cautious about classifying something as a positive. The high number of negative examples relative to the number of positive examples, and the corresponding concern of a high false positive rate, helps mitigate this potential problem. In fact, at similar levels of recall, TAN has a lower false positive rate than voting.

5 The CLP($\mathcal{BN}$) Representation

Using Bayesian classifiers to join the rules means that we will have two distinct classifiers using very different technology: a logic program (a set of rules), and a Bayes net. Some further insight may be obtained by using formalisms that combine logic and probabilities, such as CLP($\mathcal{BN}$).

CLP($\mathcal{BN}$) is based on the observation that in Datalog, missing values are represented by Skolem constants; more generally, in logic programming missing values, or existentially-quantified variables, are represented by terms built from Skolem functors. CLP($\mathcal{BN}$) represents such terms with unknown values as *constraints*. Constraints are kept in a separate *store* and can be updated as execution proceeds (ie, if we receive new evidence on a variable). Unifying a term with a constrained variable invokes a specialized *solver*. The solver is also activated before presenting the answer to a query. Syntactically, constraints are represented as terms of the form $\{C = Skolem\ with\ CPT\}$, where C is the logical variable, $Skolem$ identifies the skolem function, and CPT gives the parameters for the probability distribution.

First, we show how the Naïve Bayes net classifier can be built using CLP($\mathcal{BN}$). The value taken by the classifier is a random variable that may take the value `t` or `f` with some prior probability:

```
classifier(C) :-
       { C = classifier with p([f,t],[0.25,0.75]) }.
```

Each rule I's score V is known to depend on the classifier only:

```
rule(I,V) :-
       classifier(C),
       rule_cpt(I,P1,P2,P3,P4),
       { V = rule(I) with p([f,t],[P1,P2,P3,P4],[C]) }.
```

Rule I's score is V, which is either `f` or `t`. The value of V depends on the value of the classifier, C, according to the conditional probability table `[P1,P2,P3,P4]`. Our implementation stores the tables for each rule in a database:

```
rule_cpt(1,0.91,0.66,0.09,0.34).
rule_cpt(2,0.98,0.87,0.02,0.13).
rule_cpt(3,0.99,0.79,0.01,0.21).
rule_cpt(4,0.99,0.87,0.01,0.13).
.....
```

This fully describes the Bayes net. To actually evaluate a rule we just need to introduce the evidence given by the different rules:

```
nbayes(A,B,C) :-
        all_evidence(0,39,A,B),
        classifier(C).

all_evidence(N,N,_,_).
all_evidence(I0,N,A,B) :-
        I0 < N, I is I0+1,
        rule_evidence(I,A,B),
        all_evidence(I,N,A,B).

rule_evidence(I,A,B) :- equals(I,A,B), !, rule(I,t).
rule_evidence(I,A,B) :- rule(I,f).
```

The predicate `nbayes/3` receives a pair of individuals `A` and `B`, adds evidence from all rules, and then asks for the new probability distribution on the classifier,`C`. The predicate `all_evidence` recursively considers evidence from every rule. The predicate `rule_evidence/3` calls rule `I` on the pair `A` and `B`. If the rule succeeds, evidence from rule `I` is `t`, otherwise it adds evidence `f`.

A TAN network only differs in that a rule node may have two parents, the classifier `C` and some other node `J`. This is described in the following clause:

```
rule(I,V) :-
       rule_cpt(I,J,P1,P2,P3,P4,P5,P6,P7,P8),
       classifier(C),
       rule(J,V1),
       { V = rule(I) with p([f,t],[P1,P2,P3,P4,P5,P6,P7,P8],[C,V1]) }.
```

More complex networks can be described in a similar fashion.

CLP($\mathcal{BN}$) offers two main advantages. First, we can offer interactive access to the full classifier. Second, we gain some insight since our task now involves learning a single CLP($\mathcal{BN}$) program, where each newly induced rule will result in recomputing the probability parameters currently in the database.

6 Relationship to Other Work

Our present work fits into the popular category of using ILP for feature construction. Such work treats ILP-constructed rules as Boolean features, re-represents each example as a feature vector, and then uses a feature-vector learner to produce a final classifier. To our knowledge, the work closest to ours is by Kononenko and Pompe [12], who were the fist to apply Naïve Bayes to combine clauses. Other work in this category was by Srinivasan and King [17], for the task of predicting biological activities of molecules from their atom-and-bond structures. Some other research, especially on propositionalization of First Order Logic (FOL) [1], have been developed that convert the training sets to propositions and then apply feature vector techniques to the learning phase. This is similar to what we do; however, we first learn from FOL and then learn the network structure and parameters using the feature vectors obtained with the FOL training, resulting in much smaller feature vectors than in propositionalization.

Our paper contributes three novel points to this category of work. First, it highlights the relationship between this category of work and ensembles in ILP, because when the feature-vector learner is Naïve Bayes the learned model can be considered a weighted vote of the rules. Second, it shows that when the features are ILP-learned rules, the independence assumption in Naïve Bayes may be violated badly enough to yield a high false positive rate. This false positive rate can be brought down by permitting strong dependencies to be explicitly noted, through learning a tree-augmented Naïve Bayes net (TAN). Third, the present paper provides some early experimental evidence suggesting that a more computationally expensive full Bayes net learning algorithm may not provide added benefit in performance.

7 Conclusions

One often has to deal with erroneous and missing information in multi-relational data mining. We compare how four different approaches for combining rules learned by an ILP system perform for an application where data is subject to corruption and unobservability. We were particularly interested in Bayesian

methods because they associate a probability with each prediction, which can be thought of as the classifier's confidence in the final classification.

In our application, we obtained the best precision/recall results using a TAN network to combine rules. Precision was a major concern to us due to the high ratio of negative examples to positive examples. TAN had better precision than Naïve Bayes because it is more robust at handling high redundancy between clauses. TAN also outperformed voting in this application. Initial results for the sparse candidate algorithm show a significant increase in computation time, but no significant improvements in precision/recall.

In future work we plan to experiment with different applications and with full Bayesian networks trained using a discriminative scoring function. We also plan to further continue work based on the observation that we learn a single CLP($\mathcal{BN}$) network: this suggests that the two learning phases could be better integrated.

8 Acknowledgments

Support for this research was partially provided by U.S. Air Force grant F30602-01-2-0571. We would also like to thank the referees for their insightful comments.

References

1. E. Alphonse and C. Rouveirol. Lazy propositionalisation for relational learning. In H. W., editor, *14th European Conference on Artificial Intelligence, (ECAI'00) Berlin, Allemagne*, pages 256–260. IOS Press, 2000.
2. C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependece trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.
3. T. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2000.
4. I. Dutra, D. Page, V. Santos Costa, and J. Shavlik. An empirical evaluation of bagging in inductive logic programming. In S. Matwin and C. Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 48–65. Springer-Verlag, 2003.
5. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 148–156. Morgan Kaufman, 1996.
6. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian networks classifiers. *Machine Learning*, 29:131–163, 1997.
7. N. Friedman, I. Nachman, and D. Pe'er. Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 206–215, San Francisco, CA, 1999. Morgan Kaufmann Publishers.
8. D. Geiger. An entropy-based learning algorithm of bayesian conditional trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference (UAI-1992)*, pages 92–97, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

9. S. Hoche and S. Wrobel. Relational learning using constrained confidence-rated boosting. In C. Rouveirol and M. Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 51–64. Springer-Verlag, September 2001.
10. S. Hoche and S. Wrobel. A comparative evaluation of feature set evolution strategies for multirelational boosting. In T. Horváth and A. Yamamoto, editors, *Proceedings of the 13th International Conference on Inductive Logic Programming*, volume 2835 of *Lecture Notes in Artificial Intelligence*, pages 180–196. Springer-Verlag, 2003.
11. J. M. Kubica, A. Moore, and J. Schneider. Tractable group detection on large link data sets. In *The Third IEEE International Conference on Data Mining*, pages 573–576. IEEE Computer Society, November 2003.
12. U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 417–436. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
13. J. R. Quinlan. Boosting first-order learning. *Algorithmic Learning Theory, 7th International Workshop, Lecture Notes in Computer Science*, 1160:143–155, 1996.
14. V. Santos Costa, D. Page, M. Qazi, and J. Cussens. CLP($\mathcal{BN}$): Constraint Logic Programming for Probabilistic Knowledge. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI03)*, pages 517–524, Acapulco, Mexico, August 2003.
15. R. C. Schrag. EAGLE Y2.5 Performance Evaluation Laboratory (PE Lab) Documentation Version 1.5. Internal report, Information Extraction & Transport Inc., April 2004.
16. A. Srinivasan. *The Aleph Manual*, 2001.
17. A. Srinivasan and R. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton, editor, *Proceedings of the Sixth Inductive Logic Programming Workshop*, LNAI 1314, pages 89–104, Berlin, 1997. Springer-Verlag.

Appears in the *Proceedings of the 14th International Conference on Inductive Logic Programming* (2004)

Learning an Approximation to Inductive Logic Programming Clause Evaluation

Frank DiMaio and Jude Shavlik

Computer Sciences Department, University of Wisconsin - Madison,
1210 W. Dayton St., Madison, WI 53706
{dimaio,shavlik}@wisc.edu

Abstract. One challenge faced by many Inductive Logic Programming (ILP) systems is poor scalability to problems with large search spaces and many examples. Randomized search methods such as stochastic clause selection (SCS) and rapid random restarts (RRR) have proven somewhat successful at addressing this weakness. However, on datasets where hypothesis evaluation is computationally expensive, even these algorithms may take unreasonably long to discover a good solution. We attempt to improve the performance of these algorithms on datasets by learning an approximation to ILP hypothesis evaluation. We generate a small set of hypotheses, uniformly sampled from the space of candidate hypotheses, and evaluate this set on actual data. These hypotheses and their corresponding evaluation scores serve as training data for learning an approximate hypothesis evaluator. We outline three techniques that make use of the trained evaluation-function approximator in order to reduce the computation required during an ILP hypothesis search. We test our approximate clause evaluation algorithm using the popular ILP system Aleph. Empirical results are provided on several benchmark datasets. We show that the clause evaluation function can be accurately approximated.

1 Introduction

Inductive Logic Programming (ILP) systems [1] have been widely used in classification, data mining, and information extraction tasks. Their natural treatment of relational data, harnessing the expressive power of first-order logic, makes them useful for working with databases containing multiple relational tables. ILP systems combine background domain knowledge and categorized training data in constructing a set of rules in the form of first-order logic clauses. Formally, given a training set of positive examples E^+, negative examples E^-, and background knowledge B, all as sets of clauses in first-order logic, ILP's goal is to find a hypothesis (a set of clauses in first-order logic) h, such that

$$B \cup h \Rightarrow E^+ \qquad B \cup h \not\Rightarrow E^- \tag{1}$$

That is, given the background knowledge and the hypothesis, one can *deduce* all of the positive examples, and none of the negative examples. In real world applications, these constraints are typically relaxed, allowing h to explain *most* positive examples

and *few* negative examples. ILP systems have been successfully employed in a number of varied domains including molecular biology [2,3], engineering design [4], natural language processing [5], and software analysis [6].

One challenge many ILP systems face is scalability to large datasets with large hypothesis spaces. We define a general framework for learning a function that *estimates* the goodness of a hypothesis without looking at actual data. We suggest a number of ways in which such an approximation may be employed. One possible application eliminates poor hypotheses without wasting time evaluating them. Another uses the approximate hypothesis evaluator to guide the generation of promising new candidate hypotheses. Yet another application mines the estimator function itself for rules that can be used to invent useful predicates.

The remainder of the paper is structured as follows. Section 2 provides a background and related work on scaling up ILP. Section 3 describes construction of the hypothesis evaluation estimator. Section 4 describes in detail possible uses of such an estimator function. Section 5 shows some results of estimator learning on benchmark datasets, and Section 6 presents future research directions.

2 ILP Background and Related Work

The algorithm underlying most ILP systems is basically the same – it treats hypothesis generation as a local search in the *subsumption lattice* [7]. The subsumption lattice is constructed based on the idea of specificity of clauses. Specificity here refers to implication; a clause C is more specific than a clause S if $S \Rightarrow C$. In general, it is undecidable whether or not one clause in first-order logic *implies* another [8], so ILP systems use the weaker notion of *Plotkin's θ-subsumption*. Subsumption of candidate clauses puts a partial ordering on all clauses in hypothesis space. With this partial ordering, a lattice of clauses can be built, as in Figure 1. ILP implementations perform some type of local search over this lattice when considering candidate hypotheses.

The major distinction separating various ILP implementations is the strategy used in exploring the subsumption lattice. Algorithms fall into two main categories (with

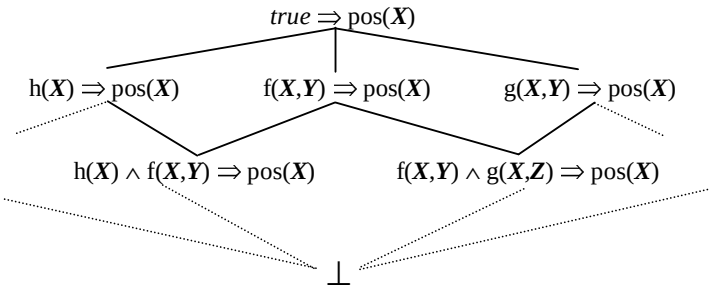


Figure 1. This illustrates an example of the subsumption lattice over which many ILP implementations search. The lattice is bounded above by *true*, and below by the bottom clause. Many ILP systems treat clause discovery as local search, moving along lattice edges.

some exceptions): general-to-specific ("top-down") [9] and specific-to-general ("bottom-up") exploration of the subsumption lattice [10]. Within these two frameworks, a variety of common local search strategies have been employed, including breadth-first search [11], depth-first search, heuristic-guided hill-climbing variants [10,11], uniform random sampling [12], rapid random restarts [13], and genetic algorithms [14]. Our work provides a general framework for increasing the speed of any ILP algorithm, regardless of the order candidate clauses are evaluated.

One challenge ILP systems face is that they scale poorly to large datasets. Srinivasan [12] investigated the performance of various ILP algorithms, and found that the running-time depends on two factors: (1) the size of the subsumption lattice and (2) the time required for clause evaluation, which in turns depends on the number of examples in the background corpus.

The first factor – the size of the subsumption lattice – mainly depends on the number of terms in a specific example's *saturation*. Saturation is used to put a lower bound on the subsumption lattice. The process is performed on a *single positive example*. Using the background knowledge, saturation constructs the *most specific, fully-ground* clause that entails the chosen example. It is constructed by applying *all possible substitutions* for variables in the background knowledge *B* with ground terms in *B*. This clause is called the chosen example's *bottom clause*, and it serves as the bottom element ($\perp$) in the subsumption lattice (Figure 1) over which ILP searches. That is, all clauses considered by ILP (in the subsumption lattice) subsume $\perp$.

As a simple example, suppose we are given background knowledge (using Prolog notation where ground atoms are denoted with an initial lowercase letter and variables are denoted with an initial uppercase letter):

`f(e,b)` `g(b,c)`
$\forall$`X,Y,Z` `f(X,Y)` $\wedge$ `g(Y,Z)` $\Rightarrow$ `h(Y)`

We are also given the current positive example, `e`.

We first begin saturation by letting all ground atoms in *H* imply *e*:

`f(e,b)` $\wedge$ `g(b,c)` $\Rightarrow$ `positive(e)`

Then we apply all possible consistent substitutions, i.e., if we make the substitutions {`e/X`, `b/Y`, `c/Z`} (using the notation {*atom/Variable*} to indicate 'atom' is being substituted for 'Variable'), we can apply the rule given in the third line of our background knowledge, that is:

`f(e,b)` $\wedge$ `g(b,c)` $\Rightarrow$ `h(b)`

Finally, combining gives us the saturation of *e*:

`f(e,b)` $\wedge$ `g(b,c)` $\wedge$ `h(b)` $\Rightarrow$ `positive(e)`

Clearly, the size of the subsumption lattice is directly related to the size of $\perp$. If we ignore multiple variablizations of a single ground literal and consider only hypotheses that contain less than *c* terms, then the size of the subsumption lattice − given a bottom clause $\perp$ − is at most $O(|\perp|^c)$ [12]. Taking into account multiple variablizations introduces an additional factor, exponential in the number of constants in the bottom clause.

The second factor – the evaluation time of a clause – is more complicated to analyze. Srinivasan simplifies the analysis by assuming that every clause can be evaluated on an example in constant time β; thus, the evaluation of a clause against the entire training set occurs in time $\beta|E| = O(|E|)$ where E is the set of training examples. An exhaustive search of the subsumption lattice for a single clause, then, takes worst-case running time $O(|\perp|^c|E|)$.

However, for most datasets clause evaluation is even worse than O(|E|). Srinivasan's work assumed that deducing each candidate hypothesis takes constant time. However, even with just one recursive rule and one background fact, deduction can be undecidable [15]. Restricting ourselves to the simpler case where function symbols are not considered (i.e., Datalog) and not allowing recursive clauses, evaluating a candidate clause against a set of ground background facts is NP-complete [16]. Most ILP datasets fall into this simpler, function-free category, where evaluation time is exponential (unless P=NP) in the number of variables, which relates to the length of the expression. In other words, a long hypothesis will take significantly longer to test against the examples in the background knowledge than will a shorter hypothesis. For many large datasets, it is precisely these long hypotheses that are most interesting. As a result, approaches to scaling up ILP [9,10] have focused upon one of these two factors: reducing the number of clauses considered, or decreasing the time spent on clause evaluations.

In reducing the number of clauses considered, the simplest techniques employ general AI search strategies, such as **A***, iterative deepening, or beam search, to reduce the number of clauses in the subsumption lattice considered. For example, using a beam reduces the worst-case running time to $O(|\perp||E|)$. However, for extremely large datasets where $|\perp|$ may be in the thousands and $|E|$ in the hundred thousands, even this may take prohibitively long.

A novel approach at reducing the number of clauses in the subsumption lattice considered has been successfully employed by Srinivasan. It uses a random sampling strategy that considers sampling n clauses from the subsumption lattice, where the value of n chosen is independent of the size of the subsumption lattice. This gives worst-case running time of $O(|E|)$ for finding a single clause. However, Srinivasan's idea only works for domains where there are a sizable number of "sufficiently good" solutions. Recent work by Zelezny *et al.* [13] has coupled random clause generation method with heuristic search using the idea of *rapid random restarts* (RRR) to explore the subsumption lattice. They repeatedly generates random clauses followed by a short local search. Rückert and Kramer [17] have also had success using stochastic search for bottom-up rule learning, outperforming GSAT and WalkSAT.

Other ILP optimizations focus instead on decreasing the time spent on clause evaluations: the $|E|$ term in ILP's running time. Several improvements to Prolog's clause evaluation function have been developed. Blockeel *et al.* [18] consider reordering candidate clauses to reduce the number of redundant queries. Santos Costa *et al.* [19] developed several techniques for intelligently reordering terms within clauses to reduce backtracking. Srinivasan [20] developed a set of techniques for working with a large number of examples that only considers using a fraction of all available examples in the learning process. Sebag and Rouveirol [21] use stochastic

matching to perform approximate inference in polynomial (as opposed to exponential) time. Maloberti and Sebag [22] provide an alternative to Prolog's SLD resolution for θ-subsumption. They instead treat θ-subsumption as a constraint satisfaction problem (CSP), then use a combination of CSP heuristics to quickly perform θ-subsumption.

Our work is distinct from all of these techniques. We describe a method for learning a function that *estimates* the clause evaluation function, which can be used in several different ways. It can reduce the evaluation time of a clause by quickly approximating the goodness of a clause, in an amount of time *independent of the number of training examples*. We can couple it with Zelezny *et al*.'s rapid random restart method in order to bias restarts toward better regions in the search space. We can use it in a manner similar to Boyan and Moore's STAGE algorithm [23] to escape local maxima in a heuristic search. Finally, we can extract hypotheses and perform predicate invention using the estimator itself.

3 Learning the Clause Evaluation Function

Heuristic approaches to exploring the subsumption lattice all make use of a scoring function to represent the *goodness* of a hypothesis at explaining the training data. Given a hypothesis (a candidate clause in first-order logic) h, a set of categorized training examples $E = \{E^+, E^-\}$, π^E_{evalfn} maps clause h to h's score on training set E under scoring metric *evalfn*:

$$\pi^E_{evalfn} : h \rightarrow \Re \qquad (2)$$

We use a multilayer, feed-forward neural network described in Section 3.1 to learn an approximate scoring function $\hat{\pi}^E_{evalfn}$. Some preliminary testing revealed that other machine learning algorithms (e.g. naïve Bayes, linear regression, C4.5) were significantly less accurate at approximating the clause evaluation function. Furthermore, a neural network with a single hidden layer is capable of approximating any bounded continuous function with arbitrarily small error [24]. We use an online training algorithm detailed in the Section 3.2 to train the neural network.

3.1 Neural Network Topology

Before constructing our clause evaluation function approximator, we need a method for encoding clauses as neural network inputs. Our encoding is based on the top-down lattice exploration used by a number of popular ILP implementations. In such implementations, a positive example is chosen at random from the training set. The chosen example is then saturated, building a bottom clause ($\perp$). Recall that this bottom clause consists of only fully ground literals. An ILP system constructs candidate hypotheses by choosing a subset of these fully-ground literals and "variablizing," replacing ground atoms with variables in a manner that replaces multiple instances of a single ground atom with a single variable (our approach does

not consider multiple – or *split* - variablizations of a single set of fully-ground literals). Approaches differ in how they select ground literals from the bottom clause.

Our neural-network inputs are comprised of a set of features derived from the candidate clause both *before* and *after* variablization. When saturating an example, each literal in that example's bottom clause is associated with an input in the neural network. This input is set to **1** if the corresponding literal in the bottom clause was used in constructing the clause, and set to **0** otherwise. Notice that there may be multiple sets of literals from the bottom clause that, when variablized, yield the same clause. This means there may be many different input representations for a single clause. However, we only use the input representation corresponding to the specific literals that *were actually chosen* when constructing the candidate clause.

Formally, let candidate clause C be chosen by selecting some subset of literals from the most-specific bottom clause $\perp_i$ for current example e_i. We treat this clause as a vector $\vec{x} = \{x_1, \ldots, x_{|\perp_i|}\}$ in $|\perp_i|$-dimensional space, with:

$$x_k = \begin{cases} 1 & \text{if ground literal k chosen in constructing C} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

This vector $\vec{x}$ is a subset of the inputs to our neural network. One important aspect of the input vector is that every possible candidate clause − that is, every clause in the subsumption lattice − has a unique input vector representation. However, the mapping does not work in the other direction: not every possible bit vector corresponds to a legal clause. In many cases, the majority of bit vectors correspond to *illegal* clauses, which contain unbound input variables. (Algorithms using the neural network to search the space of bit vectors, as in Section 4.2, need to be aware of this).

Additionally, we give each *predicate* a specific input in the network, as well. Here, we consider a vector $\vec{y}$, in which each dimension corresponds to a predicate appearing in $\perp_i$. Construction of $\vec{y}$ is based upon the number of times a particular predicate is used in a candidate clause, that is:

$$y_j = \#\text{ of ground literals in } C \text{ of predicate } j \quad (4)$$

Finally, a third set of inputs to the neural network comes from features extracted from the variablized clause C'. These features include

- **length** - number of literals in C'.
- **nvars** - number of *distinct* variables in C'.
- **nshared_vars** - number of distinct variables appearing more than once in C'.
- **avg_var_freq** - average number of times each variable appears in C'.
- **max_var_chain** - longest variable chain appearing in C', i.e., the clause `f(A):-g(A,B),h(B,C)` has max chain 3 (`A`→`B`→`C`).

The neural network consists of one (fully-connected) hidden layer and a two output units. The output units correspond to P and N, the predicted positive and negative coverage of a clause (that is, the number of examples from E^+ and E^-, respectively, deduced from the hypothesis). Given these predicted values and a scoring function, computation of the predicted output $\hat{\pi}^E_{evalfn}$ is trivial. For example, commonly used evaluation functions include *coverage* ($P-N$) and *accuracy* ($P/P+N$). Thus, we can

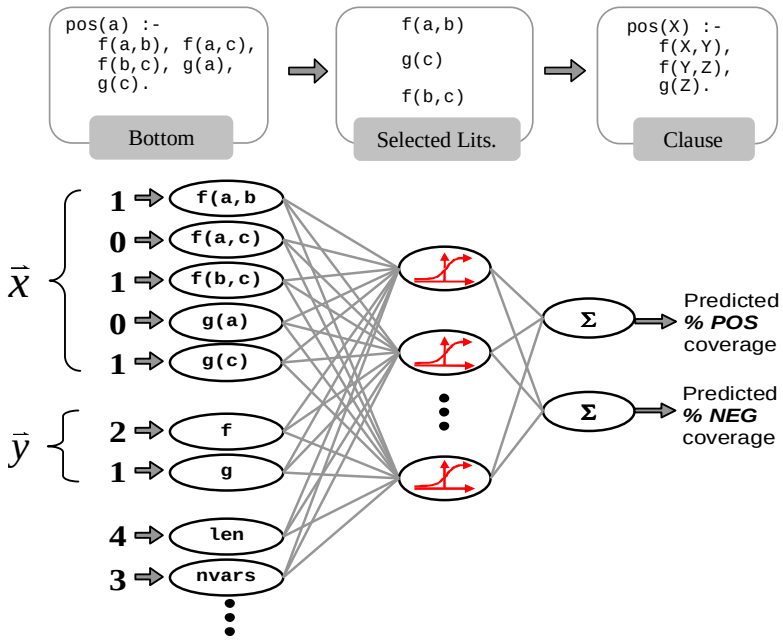


Figure 2. An overview showing the neural network's topology, and an example of input vector construction. Notice that the vector $\vec{x}$ is constructed by the literals chosen from the fully-ground bottom clause, not the candidate clause. It is quite possible for several different sets of selected literals to correspond to the same candidate clause; we only consider the set that was *actually chosen* in the clause's construction.

evaluate a clause on the neural network by converting it to the vector notation specified in Equations **(3)** and **(4)**, forward-propagating it on a trained neural network to estimate $\hat{P}$ and $\hat{N}$, and calculating $\hat{\pi}^{E}_{evalfn}$ from $\hat{P}$ and $\hat{N}$. Figure 2 presents this network topology graphically.

3.2 Online Training

The neural network's initial training makes use of Srinivasan's random uniform sampling [12]. The user specifies a burn-in length *b*, and the algorithm uniformly randomly selects *b* clauses from the space of legal clauses (up to a given maximum clause length). We evaluate these clauses on the training data, thereby creating input/output pairs for training. Using uniform sampling to generate I/O pairs ensures that the neural network approximation is reasonably accurate over the entire search space. Using other local search methods tends to bias the neural network's approximation toward some local region in the search space. Table 1a contains an overview of the algorithm used to initially train the neural network.

The methods we present in the Section 4 – that use our approximation to explore the subsumption lattice – continue to evaluate clauses (on actual data) once the

relatively short burn-in period is concluded. It seems wasteful to just throw this potential training data for the network approximation away. Our algorithm uses an *online learning algorithm* to make use of these clause evaluations – that occur as part of ILP's regular search – to improve the accuracy of the approximation. This allows us to generate a virtually unlimited number of I/O pairs for our network by simply scoring clauses on actual data.

Our online training algorithm is shown in Table 1b. When a clause is evaluated by ILP, generating an I/O pair for training our neural network, our online learning algorithm adds the pair to a cache of recently evaluated clauses. The cache typically stores 1000 to 10000 recently evaluated clauses, and, once full, elements in the cache are randomly removed to make room for incoming elements. At regular intervals (typically every 50-100 insertions) the neural network is updated by backpropagation, using the entire cache for a fixed number of epochs (typically 10). The continually changing training set, relatively short training intervals, and small number of hidden units (typically 5-10) prevent overtraining.

While the goal of our approximation is to learn an approximation of the clause evaluation function over the entire subsumption lattice, we are *especially* concerned with high accuracy of this approximation in high-scoring regions of the subsumption lattice. To ensure this accuracy, we also maintain a cache of the *best* clauses seen so

Table 1: The Neural Network burn-in training and online training algorithms. (a) The burn-in training algorithm. Given bottom clause $\perp_i$, a set of training examples *E*, and the size of the training set *trainset_size*, train a neural network to learn the clause evaluation function π^{E}_{evalfn}. We use early stopping to avoid overtraining, returning the learned network. **(b)** The online training algorithm, called for each I/O pair <*C*,{*pos*,*neg*}> that ILP generates. The algorithm keeps a cache of recent and best-scoring clauses. At some regular interval (every *arrivals_between_updates* arrivals), the algorithm updates trained network *NN* for a preset number of epochs (*epochs_per_update*). When a new arrival overflows the cache, it removes old items at random.

(a)

```
BurninTraining(⊥i, E, burnin)
  IOPairs ← ∅
  NN ← new NeuralNetwork
  minError ← +inf

  for i = 1 to burnin
    C ← rand. clause built from ⊥i
  {pos,neg} ← evaluate(evalfn, C, E)
    add <C,{pos,neg}> to IOPairs

  Split IOPairs into TrainSet and TuneSet

  for j = 1 to MAX_EPOCHS
    foreach <ex,{pos,neg}> in TrainSet
      run backprop on NN using <ex,{pos,neg}>
    error ← SSE of NN on TuneSet
    if (error < minError)
      minError ← error
    bestNN ← NN

  return bestNN
```

(b)

```
OnlineTrainingArrival(NN, <C,{pos,neg}>)
  if full(recent_cache)
    delete_random(recent_cache)
  insert <C,{pos,neg}> into recent_cache

  if score(pos,neg) > min(best_cache)
    insert <C,{pos,neg}> sorted into best_cache

  num_arrivals ← num_arrivals + 1
  if (num_arrivals = arrivals_between_updates)
    num_arrivals ← 0

    for j = 1 to epochs_per_arrival
      foreach <ex,{p,n}> in recent_cache
        run backprop on NN using <ex,{p,n}>
        foreach <ex,{p,n}> in best_cache
          run backprop on NN using <ex,{p,n}>

    return NN
```

far. This cache is typically 10% of the size of the recent-clauses cache, and when this cache is full, the lowest-scoring element is always removed to make room for incoming, higher-scoring clauses. When the neural network is updated, clauses in the best-scoring cache are also added to the training set and used to update the neural network as well.

4 Using the Clause Evaluation Approximation

This section describes three methods for using our clause approximator to scale ILP to larger datasets, and speed discovery of high-scoring hypotheses. These methods are:

(1) approximately evaluating clauses during the search of the subsumption lattice
(2) using the *evalfn* surface defined by the neural network to escape local maxima and to bias random restarts
(3) extracting hypotheses and performing predicate invention using the approximator function

4.1 Rapidly Exploring the Subsumption Lattice Using the Clause Approximator

This first method allows us to piggyback on just about any other local search method (though not stochastic methods). We perform our search in the usual manner; however, when we expand a node, instead of evaluating successor clauses on the complete set of examples, we use the neural network to compute the *approximate* clause evaluation score $\hat{\pi}^{E}_{evalfn}$. We then choose the next node to expand depending on our search strategy and the approximate scores. If this next node was approximately scored on the network, we then score it on actual data (and cache it for future training). We expand this new node and repeat the process. Recall that approximate evaluation takes O(1) running time, not the $O(|E|)$ running time required to perform the actual evaluation on the training data.

Interestingly enough, the behavior of this technique varies quite a bit depending on the search strategy employed. For a branch-and-bound search, this method serves to optimize the order in which clauses are evaluated – coupled with pruning, this could significantly reduce the total number of $O(|E|)$ real evaluations required. With **A*** search, this instead lets one "throw away" clauses that don't seem promising without wasting time evaluating them on actual data. Clauses that the neural network predicts to score poorly will never reach the font of the open list and will never be evaluated on the actual data (Note that this does break the guaranteed optimality of **A***).

Nix and Weigend have developed a technique for using a neural network to predict not only a regression value, but also to place an error bar on its prediction [25]. Using their technique, we can instead approximately score clauses, storing them in the open list with a 95% confidence bound instead of simply their predicted score. This tends to favor evaluation of clauses that the neural network cannot accurately predict – areas that should probably be thoroughly explored (but still seem promising!).

4.2 Biasing Random Restarts towards Favorable Regions of Search Space

Additionally, we can use the surface defined by the trained neural network to guide our search. *The function encoded by a neural network with fixed weights* defines a smooth surface in the space of network inputs. We can employ this neural-network designed surface in a stochastic search. For example, we can use this surface to perform "biased" rapid random restarts (hereafter referred to as biased-RRR): instead of randomly selecting literals, we perform stochastic gradient ascent on the neural-network defined surface. That is, starting from a random clause, we perform stochastic gradient ascent on this surface. The endpoint is our "random restart": the point from which we begin evaluating clauses on the actual training examples. These "guided" restarts bias search toward better regions of the search space.

One issue that arises is that the neural network contains two separate output units – one that predicts positive coverage and one that predicts negative coverage – and we want to perform gradient ascent over the surface of some scoring function that is a (possibly nonlinear) combination of the two. Fortunately, for all of the common scoring functions we can derive a simple expression relating the derivative of the scoring function to the derivative of the two output units. The derivatives of each output unit with respect to the input – $\partial P/\partial x_i$ and $\partial N/\partial x_i$ – are easily computed with a backpropagation variant (backprop computes $\partial Err_P/\partial w_{ij}$ and $\partial Err_N/\partial w_{ij}$). Table 2 summarizes these expressions for commonly used scoring functions.

An interesting variant of this approach uses the network-defined surface to escape local maxima while performing a standard ILP best-first search. We can think of this as equivalent to the biased rapid random restart above; however, instead of some

Table 2. This table expresses the gradient of several common scoring functions π_{evalfn} in terms of the gradients of the two network output units – predicted positive and predicted negative coverage. Stochastic gradient ascent uses one of these equations to compute the network-surface gradient under some scoring function. In the equations below, *P* denotes positive coverage, *N* denotes negative coverage, and *L* denotes clause length.

Scoring Function	π	Gradient Ascent Equation
compression	$P-N$	$\frac{\partial \pi}{\partial x_i}=\frac{\partial P}{\partial x_i}-\frac{\partial N}{\partial x_i}$
coverage	$P-N-L+1$	$\frac{\partial \pi}{\partial x_i}=\frac{\partial P}{\partial x_i}-\frac{\partial N}{\partial x_i}$
accuracy	$\frac{P}{P+N}$	$\frac{\partial \pi}{\partial x_i}=\frac{1}{(P+N)^2}\left(N\cdot\frac{\partial P}{\partial x_i}-P\cdot\frac{\partial N}{\partial x_i}\right)$
Laplace	$\frac{P+1}{P+N+2}$	$\frac{\partial \pi}{\partial x_i}=\frac{1}{(P+N+2)^2}\left((N+1)\cdot\frac{\partial P}{\partial x_i}-(P+1)\cdot\frac{\partial N}{\partial x_i}\right)$
entropy	$-\frac{P}{P+N}\log\left(\frac{P}{P+N}\right)-\frac{N}{P+N}\log\left(\frac{N}{P+N}\right)$	$\frac{\partial \pi}{\partial x_i}=\frac{1}{(P+N)^2}\cdot\left(N\cdot\frac{\partial P}{\partial x_i}-P\cdot\frac{\partial N}{\partial x_i}\right)\cdot\left(\ln\frac{N}{N+P}-\ln\frac{P}{N+P}\right)$
GINI	$2\cdot\frac{P}{P+N}\left(1-\frac{P}{P+N}\right)$	$\frac{\partial \pi}{\partial x_i}=\frac{2(P-N)}{(P+N)^3}\cdot\left(P\cdot\frac{\partial N}{\partial x_i}-N\cdot\frac{\partial P}{\partial x_i}\right)$

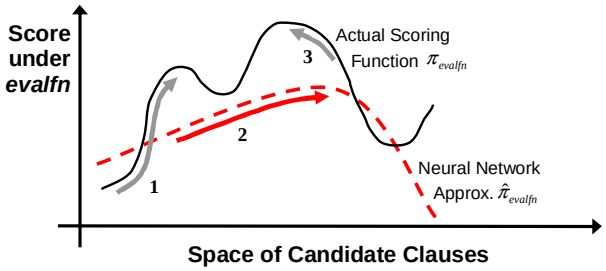


Figure 3. This graphic illustrates our algorithm using stochastic gradient ascent on the surface defined by the neural network to escaping a local minima in ILP's standard best-first search. The search alternates between periods of ILP's best-first search (1 and 3), and stochastic gradient ascent on the network-defined surface (2). The only difference between this variant and our biased-RRR search is the *starting point* of the stochastic gradient ascent. Biased-RRR begins each period of stochastic gradient ascent at a *random* point in search space.

random point, the *starting* point for our network-guided gradient ascent is the *ending* point from the previous period of ILP's standard search (on real data). That is, in this variation we rapidly alternate between brief periods of ILP's standard (best-first) search and stochastic gradient ascent on the neural-network-defined surface. This variation is illustrated in Figure 3.

This idea of intelligent rapid random restarts to escape local maxima is not a new one. Though not in the domain of ILP, Boyan and Moore's STAGE algorithm [23] use quadratic regression to approximate search "trajectories." That is, they learn a function mapping points in feature space to the endpoint of a local search starting at that point. They use this approximation to escape local maxima in a heuristic search. Their algorithm ran in less time, and reported better test-set accuracy than solutions discovered using local search alone.

4.3 Extracting Concepts from the Function Approximation

Finally, we can extract concepts from the neural network itself. Craven and Shavlik [26] have developed a method to extract a decision tree from a trained neural network. Running their algorithm on the (thresholded) trained clause-evaluation approximator would produce a theory – a set of clauses – that we could variablize and score on the actual data set.

The neural network, in fitting a nonlinear surface to the scoring function, will hopefully find pairs and triplets of terms that – while individually not helpful – lead to a highly accurate rule when combined. Two terms that share one or more variables and are connected to the same single hidden unit via a highly-weighted edge that possibly have an impact on the accuracy of the rule when taken together. Such a pair of terms is a likely candidate for terms of an invented predicate. The neural network approximation could be used to find such predicates using only one or a few seeds; then the invented predicates could be added to the background knowledge for the search over the remaining seeds' subsumption lattices.

5 Results and Discussion

This section presents our results on several benchmark datasets. We first show that the neural network is indeed capable of learning an approximation to the clause evaluation function. We then use the network in a rapid-random-restart search to bias restarts towards more promising regions of search space, as described in Section 4.2.

5.1 Benchmark Dataset Overview

We tested clause evaluation function approximation on four standard ILP benchmark datasets. The tasks included predicting mutagenic activity [27] and carcinogenic activity [28] in compounds, predicting the smuggling of nuclear and radioactive materials, and predicting metabolic activity of proteins. A brief description of the four datasets follows.

Mutagenesis. This task is concerned with predicting the *mutagenicity* of certain compounds. The ILP learner is provided background knowledge consisting of the chemical properties of 188 compounds, as well as general chemical knowledge in the form of first-order logic relations. The dataset is a popular benchmark, and explores a reasonably large search space.

Carcinogenesis. Similar to the mutagenesis task, but an inherently more difficult problem, this task's main concern is predicting *carcinogenic* activity compounds from potential carcinogenic compounds. The database for this problem consists of 332 labeled examples, of which about half are carcinogenic.

Nuclear Smuggling. This dataset, based on reports of Russian nuclear materials smuggling, is interesting in its highly-relational nature, with over 40 relational tables. The task is concerned with predicting when two smuggling events are *linked*. The dataset we use is a subset of the complete dataset, 192 examples split evenly into positive and negative examples.

Protein Metabolism. This task is taken from the gene-function prediction task of the 2001 KDD Cup challenge (www.cs.wisc.edu/~dpage/kddcup2001/). While the challenge involves learning 14 different protein functions, our sub-task is only concerned with predicting which proteins are responsible for *metabolism*. Here we also use a subset of the complete dataset, 230 examples split evenly between positives and negatives.

5.2 Learning the Clause Evaluation Function

This section details empirical evaluation of the neural network learning task. Our goal is to ascertain whether a neural network can learn the ILP clause evaluation function. To simplifying the task, in our experiments we only consider a batch learning process, not the online learning process outlined in Section 3.2.

We use the ILP system *Aleph* (web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html) to generate 10 sets of 1000 randomly sampled clauses for each of the four datasets, corresponding to 10 different positive examples

that were used in construction of the bottom clause. These 10 "seed examples" were chosen randomly. We considered a maximum clauselength c=6 for all but the Nuclear Smuggling task; we considered a larger value of c=10 for this task. Clauses were scored using a standard scoring metric, a *variant of Aleph's compression* heuristic; that is, a clause's score is given by

$$\text{score} = \frac{(\text{pos. exs. covered}) + (\text{neg. exs. covered}) - (\text{clauselength}) + 1}{(\text{total pos. exs.})} \quad (\mathbf{6})$$

Unlike Aleph's compression (which does not include the term in the denominator), we convert scores into a good range for neural networks by dividing by the total number of positive examples. This also allows comparison of scores across datasets.

For each dataset, these clauses and their corresponding scores were used to train the neural network. Using the machine learning package WEKA [29], we generated learning curves using 10-fold cross-validation. For all datasets, the neural network was constructed with 10 hidden units. The learning rate was fixed at 0.2. We added *early stopping* to WEKA to avoid overtraining. For each cross-validation fold, we set aside 33% of each training set as a tuning set. Then, after 200 epochs, we *kept the neural network that performed best on the tuning set*. WEKA's numeric feature normalization was enabled for all numeric features.

The learning curves for each of the four datasets appear in Figure 4. The "All Data" curves show the *mean* root-mean-squared (RMS) error over the 10 different sets of examples. (Section 3.4 explains the other two curves in each of these graphs.)

For all four datasets, the hypothesis evaluation function π^{E}_{evalfn} was learned with reasonable accuracy. In all four datasets, as more data is added to the training set, the neural network more accurately learns the evaluation function. It is interesting to note, however, that the number of examples required to accurately learn the approximator, and the accuracy of the final classifier varies amongst the datasets.

The absolute accuracy of the approximator varies across the datasets as well. For *protein metabolism*, the fully-trained network averages 0.005 RMS error; for *mutagenesis*, the results are an order of magnitude worse, at 0.05. Still, it seems promising that the worst performing approximator saw an RMS error of just 0.05.

So far, we have assumed no transfer of knowledge between seed examples, i.e., we learn a new neural network from scratch for *each* saturated example. However, several of the features we employ are independent of the example selected for saturation. In particular, every feature *except* the ground literals selected (the vector $\vec{x}$ described in Section 3) is instance-independent (or at least has an instance independent representation). These features can be shared when generating different rules from different seed examples, and, for all rules after the first, this allows us to bootstrap an initial classifier based on knowledge garnered from previous rules.

Consequently, we looked at the contribution of each subset of features on each of the four datasets. In particular, we wanted to see how instance-independent features contributed to the learning task. As before, we used WEKA to construct two learning curves for each dataset. These two learning curves correspond to training the network on (1) only instance-independent features, and (2) only instance-dependent features.

As Figure 4 illustrates, with the exception of *protein metabolism*, training on the instance-independent features alone did not produce as accurate a classifier as training

on the instance-dependent features alone, or on the complete set of features. Furthermore, *on all four datasets*, using the complete set of features did not produce a significantly more accurate network approximator than using the instance-dependent features alone did. This suggests that the instance-independent features are unlikely to help transfer learning for one seed example to the next seed example, and that better approaches need to be developed.

Although these graphs illustrate that we are capable of *learning* the clause evaluation function, they do not show the degree to which the function is learned. Figure 5 compares the RMS error of the network approximation to the RMS error obtained by using a random sampling of *training examples* to approximately score clauses. This provides an alternate method for computation reduction against which we compare our method. It also allows us to determine the number of evaluations the neural network is "worth." This number varies significantly across the four datasets, ranging from between 25% and 50% sampling to well beyond 90% sampling. As these are all fairly small benchmark datasets, it remains an open question how our method will compare to sampling the training examples in larger problems (with both larger hypothesis spaces as well as datasets). This includes large problems that often arise in the biological sciences and text extraction [30].

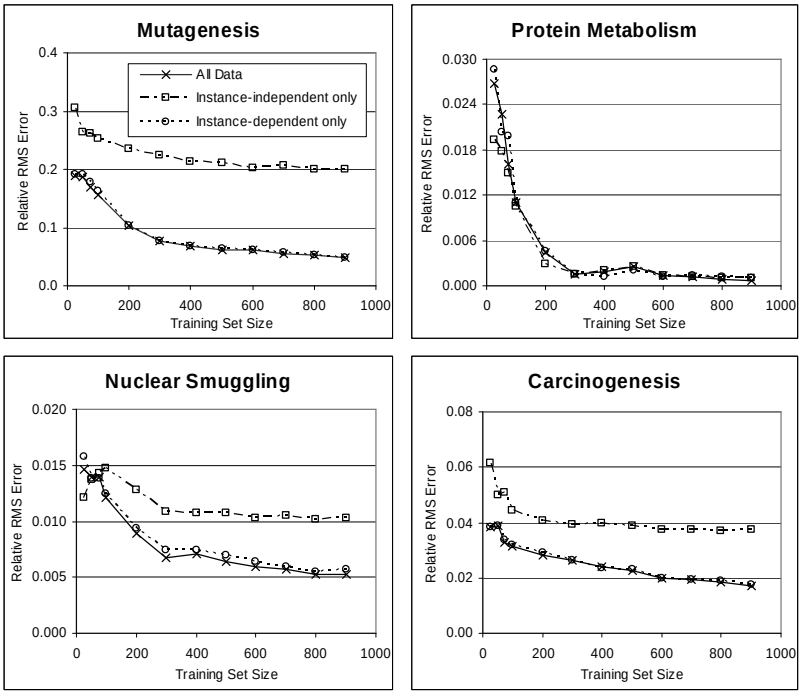


Figure 4. Learning curves showing *test-set* accuracy over four domains comparing the roles of instance-dependent versus instance-independent features. Learning curves were generated only using a subset of the complete set of features, and the results were compared to the case where all features were used to train the network.

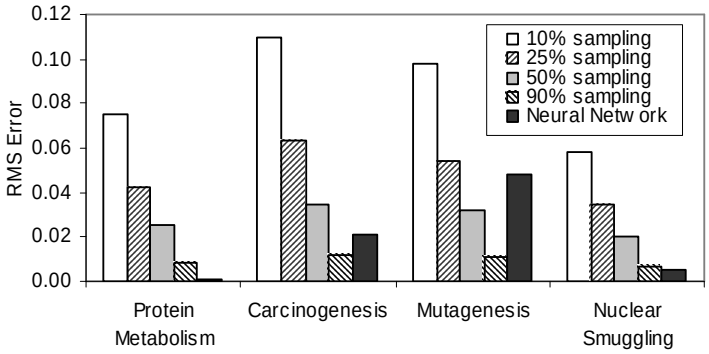


Figure 5. Comparing the RMS error of the neural-network approximation with that obtained by using a random sampling of *training examples* to approximate clauses. The error of the neural-network approximation varies widely, but in all cases does better than a 25% sampling of examples, and for two of the four datasets, does better than a 90% sampling.

5.3 Using the Evaluation Function Approximator to Guide Random Search

This section details the use of trained neural network to bias the random restarts in a rapid random restart search. Our goal here is to find the best-scoring clause in the subsumption lattice using as few clause evaluations as possible. Thus, results in this section are only concerned with maximizing some evaluation function over the *training data*. Assuming a well-designed evaluation function, this corresponds with good test-set performance.

We implemented the previous-described online learning algorithm in *Aleph*. To enable biased random restarts, we also implemented a stochastic gradient ascent algorithm. Our gradient ascent implementation, at each step, only considered flipping an input bit on or off, and did not allow flipping a bit on if the clause length was already at its maximum. The probability of a bit flip of input x_i is given by:

$$P(\text{flip } x_i) = \frac{1}{Z}\exp\left(\frac{1}{\sigma_{\mathbf{x}}^2}\cdot\frac{\partial \hat{\pi}^{E}_{evalfn}}{\partial x_i}\right)\cdot(-1)^{x_i} \qquad \textbf{(7)}$$

In this formula, σ^2 determines the "softness" of the gradient ascent. For our results, it was set such that we were 100 times more likely to flip the "best" literal than the "worst." The $(-1)^x$ term simply flips the sign of the gradient when we consider flipping a bit *off* (since this is a move in the negative direction).

In order to test the performance of our algorithm, we attempt to find the clause that maximizes the *coverage* scoring function, defined as the number of positive examples covered minus the number of negative examples covered. We used stochastic gradient ascent to bias RRR search towards with 1000 restarts and 10 steps per restart, and compare the biased-RRR versus normal RRR with the same parameters. For the biased-RRR, the "burn-in period" consisted of a single random restart and the local moves following. We report results on three of the four datasets from the previous

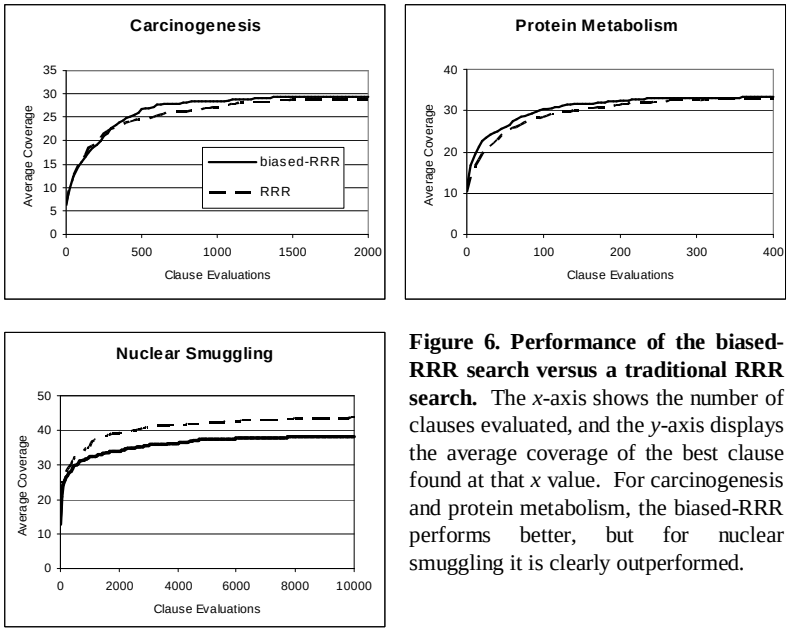


Figure 6. Performance of the biased-RRR search versus a traditional RRR search. The *x*-axis shows the number of clauses evaluated, and the *y*-axis displays the average coverage of the best clause found at that *x* value. For carcinogenesis and protein metabolism, the biased-RRR performs better, but for nuclear smuggling it is clearly outperformed.

section, omitting mutagenesis as it too quickly converges: over 80% of seeds found their best clause in the very first restart. For each dataset we explored the subsumption lattices of 100 different seed examples. Our neural network consisted of 10 hidden units. Finally, each rapid random restart began at the endpoint of the previous local search and finished after a fixed number of random steps. *Aleph* search parameters are left at default whenever possible.

Figure 6 shows the results for each of the three datasets. In each of the three graphs, the *x*-axis shows the number of clauses evaluated, and the *y*-axis shows the average coverage over all seeds of the best example found thus far. As the plots show, for two of the datasets – carcinogenesis and protein metabolism – biased-RRR found a better clause quicker than did traditional RRR. However, in the third task, nuclear smuggling, biased-RRR did worse than the default implementation. The reasons for this are unclear, as the neural network was clearly able to learn the evaluation function approximator in this domain.

6 Conclusion and Future Work

We demonstrated that the use of a neural network for clause evaluation is a useful tool for improving runtime efficiency when handling large search spaces in ILP. As ILP is confronted with increasingly larger problems, the need for methods like the ones we present grows. So far, we have treated the network learning and evaluation tasks as computationally "free" operations, which is not entirely true. However, it is true that the running time of neural network evaluation (and training) is independent

of the number of ILP examples in the dataset. This means that *given enough examples in the ILP training set*, neural-network evaluation can be made virtually free. This strategy can be used to decrease the runtime of ILP systems on large tasks.

The most pressing work that remains is implementing and evaluating the other strategies for taking advantage of the clause-evaluation approximator outlined in Sections 4.1 and 4.3. Clearly accuracy is lost in approximating the clause-evaluation function, but it is difficult to determine how it affects solutions generated by using it to quickly evaluate clauses in a typical ILP search. Another open question is whether useful information can be extracted from the trained neural network itself [26].

Also, Botta *et al.* [31] have characterized hypothesis space, discovering a critical region they have named the *phase transition*. In this critical region, the computational complexity of inference increases, and clauses generated in this region tend to have poor generalization to unseen test examples. This phase transition is a difficult region for ILP algorithms; our algorithm's performance here specifically needs exploration.

Finally, we have discussed learning the evaluation approximation in a least-squared-error sense. However, what may be more important for ILP is the relative *ranking* of candidate clauses. Thus, an approach like Caruana and Baluja's *Rankprop* algorithm [32] – an alternative to backprop concerned with correctly predicting the ranking of the output variables – may be more natural.

7 Acknowledgements

This work was supported by National Library of Medicine (NLM) grant 1T15 LM007359-01, DARPA Grant F30602-01-2-0571, United States Air Force Grant F30602-01-2-0571, and NLM grant 1R01 LM07050-01. The authors would also like to thank the UW Condor Team and the anonymous reviewers.

References

1. N. Lavrac & S. Dzeroski (1994). *Inductive Logic Programming*. Ellis Horwood.
2. R. King, S. Muggleton & M. Sternberg (1992). Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647-657.
3. A. Srinivasan, R. King, S. Muggleton & M. Sternberg (1997). The predictive toxicology evaluation challenge. *Proc. 15th Intl. Joint Conf. on Artificial Intelligence*, 1-6.
4. B. Dolsak & S. Muggleton (1991). The application of ILP to finite element mesh design. *Proc. 1st Intl. Workshop on ILP*, 225-242.
5. J. Zelle & R. Mooney (1993). Learning semantic grammars with constructive inductive logic programming. *Proc. 11th Natl. Conf. on Artificial Intelligence*, 817-822.
6. I. Bratko & M. Grobelnik (1993). Inductive learning applied to program construction and verification. *Proc. 3rd Intl. Workshop on Inductive Logic Programming*, 169-182.
7. S. Nienhuys-Cheng & R. de Wolf (1997). *Foundations of Inductive Logic Programming*. Springer-Verlag.
8. M. Schmidt-Schauss (1988). Implication of clauses is undecidable. *Theoretical Computer Science*, 59:287-296.

9. J. Quinlan (1990). Learning logical definitions from relations. *Machine Learning*, 239-266.
10. S. Muggleton & C. Feng (1990). Efficient induction of logic programs. *Proc. 1st Conf. on Algorithmic Learning Theory*, 368-381.
11. S. Muggleton (1995). Inverse Entailment and Progol. *New Generation Computing*, 13:245-286.
12. A. Srinivasan (2000). A study of two probabilistic methods for searching large spaces with ILP. *Tech. Report PRG-TR-16-00*. Oxford Univ. Computing Lab.
13. F. Zelezny, A. Srinivasan & D. Page (2002). Lattice-search runtime distributions may be heavy-tailed. *Proc. 12th Intl. Conf. on Inductive Logic Programming*, 333-345.
14. A. Giordana, L. Saitta & F. Zini (1994). Learning disjunctive concepts by means of genetic algorithms. *Proc. 11th Intl. Conf. on Machine Learning*, 96-104.
15. P. Hanschke & J. Wurtz (1993). Satisfiability of the smallest binary program. *Info. Proc. Letters*, 496:237-241.
16. E. Dantsin, T. Eiter, G. Gottlob & A. Voronkov (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33:374-425.
17. U. Rückert & S. Kramer (2003). Stochastic local search in k-term DNF learning. *Proc. 20th Intl. Conf. on Machine Learning*, 648-655.
18. H. Blockeel, L. Dehasp, B. Demoen, G. Janssens, J. Ramon & H. Vandecasteele (2002). Improving the efficiency of inductive logic programming through the use of query packs. *J. AI Research*, 16:135-166.
19. V. Santos Costa, A. Srinivasan, R. Camacho, H. Blockeel, B. Demoen, G. Janssens, J. Struyf, H. Vandecasteele & W. Van Laer (2003). Query transformations for improving the efficiency of ILP systems, *J. Machine Learning Research*, 4:465-491.
20. A. Srinivasan (1999). A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3:95-123.
21. M. Sebag & C. Rouveirol (2000). Resource-bounded relational reasoning: induction and deduction through stochastic matching. *Machine Learning*, 38:41-62.
22. J. Maloberti & M. Sebag (2001). Theta-subsumption in a constraint satisfaction perspective. *Proc. 11th Intl. Conf. on Inductive Logic Programming*, 164-178.
23. J. Boyan & A. Moore (2000). Learning evaluation functions to improve optimization by local search. *J. Machine Learning Research*, 1:77-112.
24. K. Hornik, M. Stinchcombe & H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359-366.
25. D. Nix & A. Weigend (1995). Learning local error bars for nonlinear regression. *Advances in Neural Information Processing Systems*. MIT Press.
26. M. Craven & J. Shavlik (1995). Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*. MIT Press.
27. R. King, S. Muggleton, A. Srinivasan & M. Sternberg (1996). Structure-activity relationships derived by machine learning. *PNAS*, 93:438-442.
28. A. Srinivasan, R. King, S. Muggleton & M. Sternberg (1997). Carcinogenesis predictions using ILP. *Proc. 7th Intl. Workshop on Inductive Logic Programming*, 273-287.
29. I. Witten & E. Frank (1999). *Data Mining*. Morgan Kaufmann Publishers.
30. M. Goadrich, L. Oliphant & J. Shavlik (2004). Learning ensembles of first-order clauses for recall-precision curves: a case study in biomedical information extraction. *Proc. 14th Intl. Conf. on Inductive Logic Programming*.
31. M. Botta, A. Giordana, L. Saitta & M. Sebag (2003). Relational learning as search in a critical region. *J. Machine Learning Research*, 4:431-463.
32. R. Caruana & S. Baluja (1996). Using the future to 'sort out' the present. *Advances in Neural Information Processing Systems*. MIT Press.

Appears in *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP)*. Porto, Portugal. September, 2004.

Learning Ensembles of First-Order Clauses for Recall-Precision Curves: A Case Study in Biomedical Information Extraction

Mark Goadrich, Louis Oliphant and Jude Shavlik

Department of Biostatistics and Medical Informatics and
Department of Computer Sciences,
University of Wisconsin-Madison, USA

Abstract. Many domains in the field of Inductive Logic Programming (ILP) involve highly unbalanced data. Our research has focused on Information Extraction (IE), a task that typically involves many more negative examples than positive examples. IE is the process of finding facts in unstructured text, such as biomedical journals, and putting those facts in an organized system. In particular, we have focused on learning to recognize instances of the protein-localization relationship in Medline abstracts. We view the problem as a machine-learning task: given positive and negative extractions from a training corpus of abstracts, learn a logical theory that performs well on a held-aside testing set. A common way to measure performance in these domains is to use *precision* and *recall* instead of simply using accuracy. We propose Gleaner, a randomized search method which collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an "at least N of these M clauses" thresholding method to combine the selected clauses. We compare Gleaner to ensembles of standard Aleph theories and find that Gleaner produces comparable testset results in a fraction of the training time needed for ensembles.

1 Introduction

Domains suitable for Inductive Logic Programming (ILP) can be roughly divided into two main groups. In one group, there are tasks in which each example has some inherent relational structure. One classic example of this domain is the trains dataset [20], where the goal is to discriminate between two types of trains, and the trains themselves are relational objects, having varying length and types of objects carried by each car. A more realistic example is the mutagenesis dataset [29], where the goal is to classify a chemical compound as mutagenic or not using the relational nature of the atomic structure of each chemical. ILP has proven successful in these domains by bringing the inherently relational attributes into the hypothesis space.

The other group contains tasks where examples, in addition to having a relational structure, have relations to other examples. One such domain is the

learning of friendship in social networks [2], where instead of classifying people, we try to determine the structural relationships of people based on a combination of their personal attributes and the attributes of their known friends. Another domain of this type is learning to suggest citations for scientific publications [21], where a correct citation can be a combination of data in this particular paper as well as the currently listed citations. The overall goal in these domains is to classify *links* between objects instead of the objects themselves.

Our research has focused on Information extraction (IE), the process of finding facts from unstructured text such as biomedical journals and putting those facts in an organized system. In particular, we have focused on learning multi-slot protein localization from Medline[1] abstracts, where the task is to identify *links* between phrases which correspond to a protein and the location of that particular protein in a cell. When seen as a relational data task, multi-slot IE clearly falls into the link-learning category described above.

Link-learning tasks present a number of problems to an ILP system. First, these domains tend to have a large number of objects and relations, causing a large explosion in the search space of clauses. A first approach is to sample these objects and bring the space down to a reasonable size. However, even a moderate number of objects brings about the second problem, a large skew of the data toward negative examples. Suppose in the social network domain we have 500 people, each of whom have 10 friends amongst these 500 people. This gives us 5000 positive examples, assuming that the friendship relationship is not necessarily symmetric. Our negative examples must include all other possible friendships, for $500 \times 500 - 5000 = 245,000$ negative examples, a skew of 1:49.

Information extraction is a domain that typically has unbalanced data; for example, only a very small number of phrases are protein names. Learning the relation between two entities, such as protein and location, only increases this imbalance, as the number of positive examples is now a subset of the cross-product of the entities, and the negative examples are every other pairing in the dataset.

These issues lead us away from using the standard performance measure of accuracy. Letting TP stand for true positives, FP for false positives, TN for true negatives and FN for false negatives, accuracy can be defined as $\frac{TP+TN}{TP+FP+TN+FN}$. With the positive class so small relative to the negative class, it is trivial to achieve high accuracy by labeling all test examples negative. To concentrate on the positive examples, more appropriate performance measures are *precision*, defined as $\frac{TP}{TP+FP}$, and *recall*, defined as $\frac{TP}{TP+FN}$. Precision can be seen as a measure of how accurate we are at predicting the positive class, while recall is a measure of how many of the total positives we are able to identify.

We chose to pursue IE from a machine-learning perspective. Given a set of journal abstracts manually tagged with protein-localization relationships, our goal is to learn a theory that extracts only these relations from a set of abstracts and performs well on unseen abstracts. We use five-fold cross validation, with approximately 250 positive and 120,000 negative examples in each fold. Our

[1] http://www.ncbi.nlm.nih.gov/pubmed

division of examples is not uniform because we chose to split our data into folds at the journal-abstract level (so that all the sentences in a given abstract are in the same fold), and the number of examples per abstract is variable.

We believe that ILP can be applied successfully for Information Extraction in biomedical domains as well as other link-learning tasks. ILP offers us the advantages of a straight-forward way to incorporate domain knowledge and expert advice and will produce logical clauses suitable for analysis and revision by humans to improve performance. We use Aleph [27], a mature ILP system, to learn first-order clauses.

The standard approach to ILP is to learn clauses sequentially until almost all of the positive examples are covered by at least one clause, thus creating a theory. By itself, an individual theory will produce one value for precision and recall, at least if one uses the standard logical approach of disjunction to combine the clauses in a theory. A more useful evaluation would be to create a recall-precision curve, which illustrates the trade-off between these two measurements. One way to create a recall-precision curve from a theory containing M clauses is to require that *at least* N of the clauses are satisfied. By varying N from 1 to M, one can obtain a variety of points in the recall-precision curve [10]. However, ILP systems have not traditionally been designed to produce recall-precision curves, and it is likely that specially designed algorithms will do better than simply counting the number of clauses that are satisfied by a given example.

To address the goal of efficiently producing good recall-precision curves with ILP, we propose the Gleaner algorithm. Gleaner is a randomized search method that collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an "at least N of these M clauses" thresholding method to combine the selected clauses. We compare Gleaner to ensembles of standard Aleph theories [11]. We find that Gleaner produces comparable results in a fraction of the training time needed for Aleph ensembles. These smaller theories will also reduce classification time, an important consideration when working with large domains.

2 Biomedical Information Extraction

Information Extraction (IE) is the process of scanning plain text files for objects of interest and facts about these objects. As a learning task, IE is defined as: given information in unstructured text documents, extract the relevant objects and relationships between them. There are two main IE tasks, Named Entity Recognition (NER) and Multi-Slot Extractions. NER can be seen as identifying a single type of object, for example the name of an individual, corporation, gene, or weapon. Successful rule-based approaches for named-entity IE include Rapier [8], a system which learns clauses with the format *prefix, extraction, postfix*, and Boosted Wrapper Induction (BWI) [14], a method for boosting weak rule-based classifiers of extraction boundaries into a powerful extraction method. BWI has been further examined by Kauchak et al. [17] showing results with high recall and high precision on a wide variety of tasks.

"We suggest that SMF1 and SMF2 are mitochondrial membrane proteins that influence PEP-dependent protein import, possibly at the step of protein translocation."

```
protein_location(SMF1, mitochondrial)
protein_location(SMF2, mitochondrial)
```

Fig. 1. Sample Sentence with its Correct Extractions

Multi-slot extraction builds upon the objects found in NER, and looks for a relationship between these items in the text, some examples being a parent-child relationship between individuals, the CEO of a particular company, or the interaction of two proteins in a cell. Multi-slot extraction is typically much harder; not only must the objects of the relation be identified, but also the semantic relationship between these two objects.

Recently, biomedical journal articles have been a major source of interest in the IE community for a number of reasons: the amount of data available is enormous, the objects, proteins and genes, do not have standard naming conventions, and there is a definite interest from biomedical practitioners to quickly find relevant information [3, 26]. Biomedical journals also contain highly domain-specific language, as seen in Figure 1.

Previous machine-learning work in the biomedical multi-slot domain includes a number of different approaches. Ray and Craven [23] use a Hidden Markov Model (HMM) modified to include part of speech tagging, and analyze their method on protein localization, genetic disorder and protein-protein interaction tasks. For the same datasets, Eliassi-Rad and Shavlik [13] implemented a neural network for IE primed with domain-specific prior knowledge. Aitken [1] uses FOIL to perform ILP, working with a closed ontology of entities, while Brunescu et al. [7] propose the use of ELCS, a bottom up approach to finding protein interactions with rule templates for sentences. Brunescu et al. have also extended Rapier and BWI to handle multi-slot extractions.

2.1 Data Labeling

In this paper, we focus on one particular dataset, learning the location of yeast proteins in a cell as illustrated in Figure 1. Our testbed comes from Ray and Craven [23]. The data consist of 7,245 sentences from 871 abstracts found in the Medline database, and contains 1,200 relations. In the original dataset, the labeling was performed semi-automatically, in order to avoid the laborious task of labeling by a human. Protein localizations were gathered from the Yeast Protein Database (YPD), and sentences which contained instances of both a protein and location pair were marked as positive by a computer program.

In our early exploration of the dataset, we found that there were a significant number of false positives that looked like true positives but were apparently missed by the automated labeling algorithm. Also, some of the labelings were

ambiguous at best, finding both parts of a positive protein localization, whereas the human-judged semantics of the sentence did not involve localization. In addition, by using this labeling scheme, we did not have data on all yeast proteins in the corpus, only those listed in YPD. Because of these issues, we decided to relabel the dataset by hand. We were assisted in this effort by Soumya Ray.

To label the positive examples, we manually performed both protein and location named-entity labeling and relational labeling. Our labeling standards differ from those used by other groups [16], as our task is to extract the locations of yeast proteins. If there was any disagreement among the labelers, we did not tag the protein or location, to make sure our training set was as precise as possible at the expense of some recall.

For the protein labeling, we strove to be specific rather than general, and only labeled those words that directly refered to a protein or gene molecule. This included gene names such as "SMF1", protein names like "fet3p" and full chemical names of enzymes, such as "qh2-cytochome c reductase". Therefore, while we would label SEC53 from "SEC53 mutant", we did not label "isp4delta" or "rrp1-1" as these gene products are defective and would not give rise to a functioning protein molecule. We did not label protein families such as "hsp70" unless it was an adjective to a protein, as in "hsp70 dnaK". Fusion proteins, such as when a gene is combined with a fluorescent tag, were labeled as proteins. Protein complexes, antibodies and open reading frames were never labeled as positive protein examples. Also, only proteins that are known to exist in yeast were labeled, not those which were found in other species, since our dataset dealt with the localization of yeast proteins.

Labeling the location words was much more direct. We used a list of known cellular locations listed in an introductory cellular biology text book, including locations and abbreviations such as "cytoskeleton", "membrane", "lumen", "ER", "npc", "bud", etc. Also labeled were location adjectives, such as "nucleoporin" and "ribosomal".

To determine if there was a relationship between any tagged proteins and tagged locations, we used three classifications: clear, ambiguous, or co-occurrence. Relationships directly implied by the text, as in `protein_location(YRB1p, cytosol)` from the sentence "YRB1p is located in the cytosol," were classified as *clear*, while those relationships where the protein location was implied rather than stated, such as `protein_location(LIP5, mitochondrial)` from the sentence "LIP5 mutants undergo a high frequency of mitochondrial DNA deletions," were labeled as *ambiguous*. The correct classification was agreed upon by all three labelers. For our experiments, we used the *clear* category as positive examples, and all other phrase pairings as negative examples. A future goal is to improve our manual-labeling interface.

2.2 Background Knowledge

Instead of the standard feature-vector machine learning setup, ILP uses logical relations to describe the data. Algorithms attempt to construct logical clauses based on this background structure that will separate positive and negative

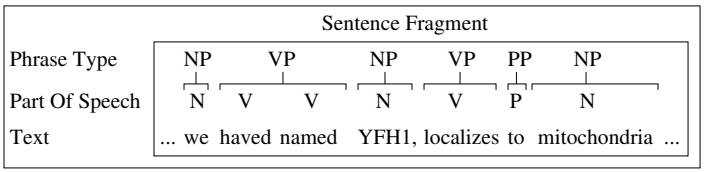


Fig. 2. Sample Sentence Parse from Sundance Sentence Analyzer (N=noun, V=verb, P=preposition or phrase)

examples. For our information extraction task, we construct background knowledge from sentence structure, statistical word frequency, lexical properties, and biomedical dictionaries.

Our first set of relations comes from the sentence structure. We use the Sundance sentence parser [24] to automatically derive a parse tree for all sentences in our dataset and the part-of-speech for all words and phrases of the tree. This tree is then flattened to some degree, so that there are no nested phrases; all phrases have the sentence as the root, and therefore all words are only members of one phrase. Figure 2 shows an example sentence parse.

Each word, phrase, and sentence is given a unique identifier based on its ordering within the given abstract. This allows us to create relations between sentences, phrases and words not based on the actual text of the document but on its structure, such as `sentence_child`, `phrase_previous` and `word_next` about the tree structure and sequence of words, and relations like `nounPhrase`, `article`, and `verb` to describe the sentence structure. To include the actual text of the sentence in our background knowledge, the predicate `word_ID_to_string` maps these identifiers to the words. In addition, the words of the sentence are stemmed using the Porter stemmer [22], and currently we only use the stemmed version of words.

Another group of background relations comes from looking at the frequency of words appearing in the target phrases in the training set. This is done on a per-fold basis to prevent learning from the test set. For example, the words "body", "npc", and "membrane" are at least 10 times more likely to appear in location phrases than in phrases in general in training set 1. We created predicates for several gradations from 2 times to 10 times the general word frequency across all abstracts in a given training set. These gradations are calculated for both arguments–protein and location–as well as for words that appear more frequently in between the two arguments or before or after them. We create semantic classes, consisting of these high frequency words. These semantic classes are then used to mark up all occurrences of these words in a given training and testing set.

A third source of background knowledge is derived from the lexical properties of each word. `Alphanumeric` words contain both numbers and alphabetic characters, whereas `alphabetic` words have only alphabetic characters. Other lexical and morphological features include `singleChar`, `hyphenated` and `capitalized`. Also, words are classified as `novelWord` if they do not appear in the standard `/usr/dict/words` dictionary in UNIX.

Sentence Structure Predicates
phrase_after(Phrase1,Phrase2)
phrase_contains_specific_word(Phrase,Word,WordString)

Statistical Word Frequency Predicates
phrase_contains_2x_word(Phrase,Argument)
phrase_contains_no_between_halfX_word(Phrase,Argument,PartOfSpeech)

Lexical Properties Predicates
alphabetic(Word)
few_wordPOS_in_sentence(Sentence,PartOfSpeech)

Biomedical Dictionaries Predicates
phrase_contains_mesh_term(Phrase,Term,StemmedTerm)
phrase_contains_go_term(Phrase,Term,StemmedTerm)

Fig. 3. Sample Predicates used in our Information Extraction Task

Finally, we incorporate semantic knowledge about biology and medicine into our background relations, such as the Medical Subject Headings (MeSH)[2], the Gene Ontology (GO)[3], and the Online Medical Dictionary[4]. As in sentence structure, we have simplified these hierarchies to only be one level. We have picked three categories from MeSH (protein, peptide and cellular structure), the cellular-localization category from GO, and the cellular-biology category from the Online Medical Dictionary, and have labeled phrases with these predicates if any of the words in the given phrase match any words in the category.

Sentence structure predicates like `word_before` and `phrase_after` are added allowing navigation around the parse tree. Phrases are also tagged as being the first or last phrase in the sentence, likewise for words. The length of phrases is calculated and explicitly turned into a predicate, as well as the length (by words and phrases) of sentences. Also, phrases are classified as short, medium or long. An additional piece of useful information is the predicate `different_phrases`, which is true when its arguments are distinct phrases.

Lexical predicates are augmented to make them more applicable to the phrase level. If a phrase contains an alphabetic word, the phrase is given the predicate `phrase_contains_alphabetic_word(A)`. Similarly phrases with specific words are marked with `phrase_contains_specific_word(A, ''lumen'')`. This is the equivalent of adding both `phrase_child(A,B)`, `word_ID_to_string(B, ''lumen'')` at once. These predicates are also created for pairs and triplets of words, so we can assert that a phrase has the word "golgi" labeled as a noun all in one search step.

[2] http://www.nlm.nih.gov/mesh/meshhome.html
[3] http://www.geneontology.org/
[4] http://cancerweb.ncl.ac.uk/omd/

Finally, predicates are added to denote the ordering between the phrases. `Target_arg1_before_target_arg2` asserts that the protein phrase occurs before the location phrase, similarly for `target_arg2_before_target_arg1`. Also created are `adjacent_target_args` (which is true when the protein and location phrases are adjacent to each other in the sentence), and `identical_target_args` (which says the same noun phrase contains both the protein and its location), as well as the count of phrases before and after the target arguments. A list of our predicate categories and some sample predicates are found in Figure 3. Overall, we have defined 251 predicates for use in describing the training examples.

2.3 Unbalanced Data Filtering

As previously mentioned, one of the difficulties we face with this domain is the large number of possible examples we must consider. Within each sentence, we need to examine each pair of phrases. With only a few positive examples, our positive:negative ratio is 1:600, leading to severely unbalanced data.

For this domain, we use prior knowledge to help reduce the number of false positive examples. We observe that 95% of our positive relations contain only noun phrases, while the overall ratio is 26%, and use this to limit the size of our training data to only those candidate extractions where both arguments are noun phrases. This reduces the positive:negative ratio in our data to 1:158. We must necessarily keep track of all missed positive in the testing set, those that have at most one non-noun phrase, and record them as false negatives in our recall-precision results.

To further reduce the positive:negative ratio we randomly under-sample the negatives, retaining only a fourth during training. This allows for faster clause learning. Future work includes selecting the "close" negative examples to use during training rather than randomly selecting them.

3 Aleph

Aleph [27], is a top-down ILP covering algorithm developed at Oxford University, UK. It is written completely in Prolog and is open source. As input, Aleph takes background information in the form of predicates, a list of modes declaring how these predicates can be chained together, and a designation of one predicate as the "head" predicate to be learned. Also required are lists of positive and negative examples of the head predicate.

As a high-level overview, Aleph generates clauses for the positive examples by picking a random example to be a seed. This example is saturated to create the bottom clause, i.e. every relation in the background knowledge that can be reached from this example. The bottom clause becomes the possible search space for clauses. Aleph heuristically searches through the space of possible clauses until the "best" clause is found or time runs out. The standard way to use Aleph is to combine these learned clauses into a theory when enough clauses are learned to cover almost all positive training examples.

Aleph is a very flexible ILP system with a wide variety of learning parameters available for modification. Some of the parameters we utilized were:

minimum accuracy. We can place a lower bound on the accuracy of all clauses learned by our system. This is only the accuracy of the clause on the examples covered by it, in other words, precision.

minimum positives. To prevent Aleph from learning narrow clauses, ones which only cover a few examples, we can specify that each acceptable clause must cover at least a certain number of positives.

clause length. The size of a particular clause can be constrained using clause length. By limiting the length, we can explore a wider breadth of clauses and prevent clauses from becoming too specific.

search strategy. As Aleph uses search to find good clauses, the type of search is a parameter. These include the standard search methods of breadth-first search, depth-first search, iterative beam search, iterative deepening, as well as heuristic methods requiring an evaluation function.

evaluation function. There are many ways to calculate the value of a node for further exploration. The most common heuristic used in ILP is *coverage*. This is defined as the number of positives covered by the clause minus the number of negatives ($TP - FP$). A very similar heuristic is *compression*, which is coverage minus the length of the clause ($TP-FP-L$). Since we are working within domains to generate precision/recall curves, we also explored as our heuristic-search's evaluation function (a) $precision \times recall$, and (b) the F1 measure, which is ($\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$). To improve clause quality and correct accuracy estimates for clauses that cover a small number of examples, one can also use the Laplace estimate, ($\frac{TP+1}{TP+FP+2}$).

coverage in tune set. To encourage our clauses to be more general, we added a parameter to Aleph requiring each recorded clause to have some small positive coverage in the tuneset. We believe this will help our clauses on the unseen examples in the test set.

4 Gleaner

Since our biomedical IE task is a link-learning task, we need to evaluate the success of our methods using precision and recall. In order to rapidly produce good recall-precision curves, we have developed Gleaner, a two-stage algorithm to (1) learn a broad spectrum of clauses and (2) then combine them into a thresholded disjunctive clause aimed at maximizing precision for a particular choice of recall. Our algorithm is summarized in Figure 4.

Our first stage of Gleaner learns a wide spectrum of clauses. We have Aleph search for clauses using K seed examples. We diversify the search by first uniformly dividing the recall dimension into B equal sized bins, for example, $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1]$. For each seed, we consider up to N possible clauses using a random local-search method. As these clauses are generated, we compute the recall of each clause and determine into which bin the clause falls.

```
Create B recall bins, uniformly dividing the range [0,1]
For i = 1 to K
    Pick a seed example to generate bottom clause
    Use Random Local Search to find clauses
    After each generation of a new clause r
        Find the recall bin b_k for r
        If the Precision × Recall of r is best yet
            Store r in b_k
For each bin b
    Find L_b ∈ [1, K] on trainset such that
        recall of "At least L of K clauses match examples" ≈ recall for this bin
Find precision and recall of testset using each bin's "at least L of K" decision process
```

Fig. 4. Gleaner Algorithm

Each bin keep tracks of the highest precision clause learned in that bin so far and will be replaced when a more precise clause is found (actually, rather than finding the highest precision clause within each bin, we save the clause whose product of precision and recall is highest among those clauses falling into this recall bin). At the end of this search process, there will be B clauses collected for each seed and K seed examples for a total of $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed).

To perform random local search, we considered four search methods, Rapid Random Restart (RRR), Stochastic Clause Selection (SCS), GSAT, and WalkSAT. SCS randomly picks clauses which are subsets of the bottom clause according to the distribution of clauses based on length. SCS has a hard time finding high quality clauses and is biased to select long clauses due to the heavy-tailed distribution of clause lengths. GSAT selects an initial clause at random and then chooses to either add or remove a randomly selected literal if the new clause is "better" according to the evaluation function; WalkSAT modifies GSAT by allowing a certain percent of "bad" moves. RRR works similarly to GSAT and WalkSAT in the initial clause selection, but only refines clauses by adding predicates (using best first search), restarting with a new clause after a specified number of evaluations. GSAT and WalkSAT occasionally make "downhill" moves in the search space, while RRR does not, and due to the interal workings of Aleph, adding predicates to a clause is much more efficient than removing them. We found that RRR both takes less time and produces higher quality clauses than the other methods, and we use it as Gleaner's search method in the remainder of this article.

The second stage takes place once we have gathered our clauses using random search. We need a way to combine these clauses into a single precision/recall point for each bin. We could choose the best clause collected from each bin, however this is likely to have poor generalization to the test set, especially for the low-recall bins. If we classify an example as positive only if it matches *all* K clauses collected for a bin, we obtain high precision, but our recall will be dras-

tically reduced. Alternatively, if we classify an example as positive if it matches *any* of our K clauses, we will probably have a theory with high recall but low precision. Instead, we need to find a balance between these two extremes, and classify examples to be positive if they are covered by a large enough subset of clauses. *Our hypothesis is that this method will produce a theory with about the same recall as the bin (by construction), but higher precision than any one clause, since we require that an example satisfy multiple clauses (assuming $L > 1$).*

Gleaner combines the clauses in each bin to create one large thresholded disjunctive clause, of the form "At least L of these K clauses must cover an example in order to classify it as a positive." We want this clause to have about the same recall as as that of the clauses in the bin (so that we cover the full range of possible recalls), thus we need to find the best threshold L for each bin. We can find this L on the training set for each bin by starting with $L = K$ and incrementally lowering the threshold to increase recall. We stop when any lower L would increase the distance between the recall of the best L of K clause and our desired recall. With this L, we now evaluate our disjunctive clause on the testset and record the precision and recall. We will end up with B precision/recall points, one for each bin, that span the recall-precision curve.

5 Ensembles in ILP

Bagging [6] is a popular ensemble approach to machine learning where multiple classifiers are trained using different subsamples of the training data. These classifiers then vote on the classification of testset examples, usually with the majority class being selected as the output classification. How they vote is user-dependent, with some common schemes being equal voting or weighted according to the tuneset accuracy of each voter. The main idea of bagging is that it will produce diverse classifiers that make their mistakes in different regions of the input space; when their votes are combined, prediction errors will be reduced.

The use of bagging for ILP has been previously investigated by Dutra et al. [11] where they demonstrate bagging to be helpful for modest improvements in accuracy as well as a straight-forward way to calculate the confidence of a particular example. We use their "random seeds" approach for creating ensembles. This approach, which Dutra et al. showed to have essentially equivalent predictive accuracy as bagging, produces diversity in its learned models by starting each run of its underlying ILP system with a different "seed" example.

We compare our Gleaner approach to that of using "random seeds" in Aleph. In this experimental control, we call Aleph N times and have it create N theories (i.e., sets of clauses that cover most of the positive training examples and few of the negative ones). To create a recall-precision curve from these N theories, we simply classify an example as positive if at least K of the theories classify it as positive; varying L from 1 to N produces a family of ensembles, and each of these ensembles produces a point on a recall-precision curve.

Aleph involves a large number of parameters, and we use the train and test sets to choose a good set (since this is the experimental control against which we

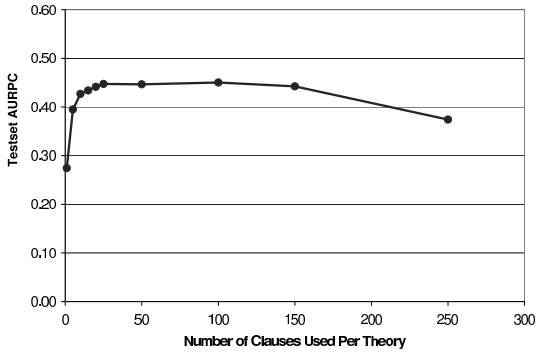


Fig. 5. Area Under the Recall-Precision Curve for 100 Aleph Ensembles With Varying Number of Clauses

compare our Gleaner system, it is "fair" to use the testset to tune parameters). We compare several different evaluation functions for judging clauses: Laplace (which essentially measures accuracy, but corrects for small coverage), coverage (the number of positive examples covered minus the number of negatives covered), $precision \times recall$, and $F1$ (the harmonic mean of precision and recall; F1 is the most commonly used performance measure in information extraction). We consider two settings for minimum accuracy for learned clauses: 0.75 and 0.90. We require all clauses to at least cover seven positive examples and to be no longer than ten terms (the same settings we use for random sampling of the hypothesis space in our Gleaner approach). We limit the number of clauses considered to 100 thousand and we also limit the number of reductions to 100 million (using the `call_counting` predicate available in YAP Prolog[5]).

We obtained our best area under the recall-precision curve using Laplace as the evaluation function and a minimum clause accuracy of 0.75. (Under this setting, the average number of clauses considered per constructed theory is approximately 35,000.)

One new finding we encountered that was not reported by Dutra et al. is that it is better to limit the size of theories. Figure 5 plots the area under the recall-precision curve (AURPC) as a function of the maximum number of clauses we allow in the learned theories. Running Aleph to its normal completion given the above parameters leads to theories containing 271 clauses on average. However, if we limit this to the first C clauses, the AURPC can be drastically better. The likely reason for this is that larger theories have less diversity amongst themselves than do smaller ones, and diversity is the key to ensembles [12]. A nice side-effect of limiting theory size is that the runtime of individual Aleph executions is substantially reduced.

In the next section, where we evaluate our Gleaner algorithm, we limit theory size in our "ensemble of Aleph theories" approach to 50 clauses, since as seen in Figure 5, testset AURPC has essentially peaked by then. In that section's

[5] http://www.ncc.up.pt/~vsc/Yap/yap.html

experiments we do vary the size of the ensemble (i.e., number of theories) and the number of clauses in each theory, in order to see the impact on AURPC as a function of the amount of time spent training.

While we are from having considered all possible parameters settings and algorithm designs with which one could use Aleph to create an ensemble of theories, we have evaluated a substantial number of variants and feel that our chosen settings provide a satisfactory experiment control against which to compare our new algorithm, Gleaner.

6 Results

For our experiments, we divided the protein localization data into five folds, equally divided at the journal-abstract level. Each training set consisted of three folds, with one fold held aside for tuning and another for testing. For our current experiments we only use the tuning set minimally, requiring each clause learned on the training set to cover at least two positive examples in the tuning set.

To evaluate the performance of our algorithms, we use recall-precision curves [19], or more precisely, we use the Area Under the Recall-Precision Curve (AURPC) to gather a single score for each algorithm. AUC has traditionally been used to analyze ROC curves [5], which plot the true positive rate versus the false positive rate. To calculate the AURPC, we first standardize our recall-precision curves to always cover the full range of recall values and then interpolate between the threshold points. From the first threshold point, which we designate (R_{first}, P_{first}), the curve is extended horizontally to the point $(0, P_{first})$, since we could randomly discard a fraction, f, of the extracted relations and expect the same precision on the remaining examples; the setting of f would determine the recall. An ending point of $(1, \frac{Pos}{Neg})$ can always be found by calling everything a positive example. This will give us a closed curve extending from 0 to 1 along the recall dimension.

For any two points A and B in a recall-precision curve, we must interpolate between their true positive (TP) and false positive (FP) counts in order to calculate the area. To do this, we create new points for each of $TP_A + 1, TP_A + 2, ..., TP_B - 1$, increasing the false positives for each new point by $\frac{FP_B - FP_A}{TP_B - TP_A}$. Interpolation for the recall-precision curve is different than for an ROC curve; whereas the ROC interpolation would be a linear connection between the two points, in recall-precision space the connection can be curved, depending the actual number of positive and negative examples covered by each point. The curve is especially pronounced when two points are far away in recall and precision. Consider a curve constructed from a single point of $(0.02, 1)$, and extended to the endpoints of $(0, 1)$ and $(1, 0.008)$ as described above (for this example, our dataset contains 433 positives and 56,164 negatives). Interpolating as we have described, would produce an AURPC of 0.031; a linear connection would overestimate with an AURPC of 0.50 (Figure 8 shows this graphically).

A sample clause found by Gleaner is shown in Figure 6. We can see for our dataset that it is important to require the protein phrase to contain

```
protein_location(P,L,S) :-
      first_word_in_phrase(L,A),
      phrase_after(L,_),
      target_arg1_before_target_arg2(P,L,S),
      after_both_target_phrases(S,B),
      phrase_contains_some_marked_up_location(L,_),
      few_POS_in_phrase(P,alphanumeric),
      few_wordPOS_in_sentence(S,alphanumeric),
      phrase_contains_no_between_halfX_word(B,between_arg1_and_arg2,verb),
      phrase_contains_some_art(L,A).
```

where P is the protein phrase, L is the location phrase, S is the sentence, and '_' indicates variables that only appear once in the clause.

Positive Extraction
"NPL3 encodes a nuclear protein with an RNA recognition motif and similarities to a family of proteins involved in RNA metabolism."
`protein_location('NPL3', 'a nuclear protein')`

Negative Extraction (i.e., a false positive)
"Subcellular fractionation studies further demonstrate that the 1455 amino acid Vps15p is peripherally associated with the cytoplasmic face of a late Golgi or vesicle compartment."
`protein_location('the 1455 amino acid Vps15p', 'the cytoplasmic face')`

Fig. 6. Sample Clause with 29% Recall and 34% Precision on Testset 1

`alphanumeric` words. Also important for this clause is the sentence structure, requiring that the protein phrase comes before the location phrase, and that the location phrase is not the last phrase in the sentence.

Our Aleph-based method for producing ensembles has two parameters that we vary: N, the number of theories (i.e., the size of the ensemble), and C, the number of clauses per ensemble. To produce ensemble points in Figure 7, we choose N from {10, 25, 50, 75, 100} and C from {1, 5, 10, 15, 20, 25, 50}, producing 20 combinations for each fold.

For the parameters of Gleaner, we used 20 recall bins and 100 seed examples to collect 2,000 clauses total. We told RRR to construct 1,000 clauses before restarting with a new random clause. We generate AURPC data points for Gleaner by choosing the number of seed examples from {25, 50, 75, 100}, and using the intervals of {1K, 10K, 25K, 50K, 100K, 250K, 500K} for the number of candidate clauses generated per seed.

The results of our comparison are found in Figure 7; the points are averaged over all five folds. Note this graph has a logarithmic scale in the number of clauses generated. We see that Gleaner can find comparable AURPC numbers using two orders of magnitude fewer clauses. It is interesting to note that the

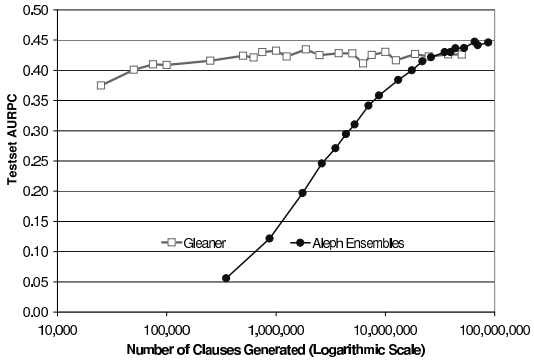


Fig. 7. Comparison of AURPC from Gleaner and Aleph Ensembles by Varying Number of Clauses Generated

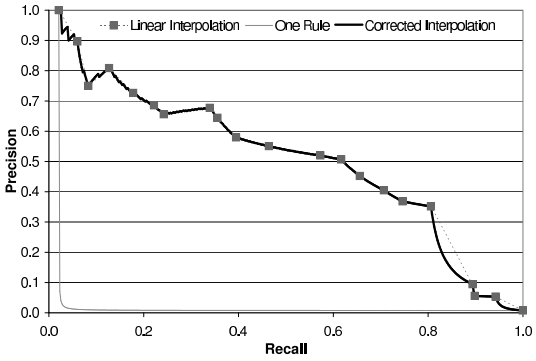


Fig. 8. A Sample Gleaner Recall-Precision Curve From Fold 5

Gleaner curve is very consistent across the number of clauses allowed, while the ensemble method increases when more clauses are considered. It is a topic of future work to devise a new version of Gleaner that is able to better utilize additional candidate clauses.

In Figure 8, we show one of the better recall-precision curve produced by Gleaner using 10,000 candidate clauses per seed and 100 seed examples (on fold 5). For comparison, we also show the one-point interpolation curve mentioned above. Gleaner's "L of K" clauses theoretically should produce higher precision than individual rules with the same recall, as long as coverage of positives is greater than coverage of negatives. In practice, our clauses are not as independent as we would like, especially in the high-recall bins, with many of the learned clauses being identical. This overlap degrades the performance.

7 Conclusions and Future Work

Multi-Slot Information Extraction is a an appealing challenge task for ILP, due to its large amount of examples and background knowledge, as well as the substantial skew of examples. We have developed a method called Gleaner, which gathers a wide spectrum of clauses and combines them within bins based on recall using an "at least N of these M clauses" thresholding method.

We find that Aleph ensembles can perform well when using early stopping (i.e., only learning a dozen or two rules); however, Aleph ensembles suffer when allotted a limited amount of time to create multiple theories. Our method of Gleaner results in similar curves to Aleph ensembles, and outperforms ensembles when both are only allowed to evaluate a limited number of clauses. There are not many large, heavily skewed datasets available for ILP research, and we believe this information-extraction task will provide a useful testbed for further ILP research. To aid in ILP research this dataset is being made available at our website (see Acknowledgements).

There are a number of approaches relating to the combination of learned clauses to produce a confidence measure, as opposed to combining multiple theories as in bagging or Gleaner. Propositionalization of the feature space has been examined by Lavrac et al. [18], which allows for any propositional learner that generates confidence measures to be used. Similarly, Srinivasan [28] investigated using ILP as a feature construction tool for propositional learners, namely linear regression. Craven and Slatterly [10] use a logical setup combined with Naive Bayes classifiers for IE and generate recall-precision curves with their resulting theories. We plan to compare these within-theory ensemble methods to the multiple theory ensemble methods and to Gleaner.

In this same vein, we see the use of boosting in ILP [15] as another alternative method to searching for clauses and learning how to combine them in one single step. Recent work has shown that a RankBoost, a variant of boosting, directly optimizes the area under the ROC curve [9]. We believe that a similar optimization of the area under the recall-precision curve can be achieved, and plan to implement this algorithm in Aleph for comparison to Gleaner.

We noticed that many of our learned clauses are focused on learning the individual entities of the relation, in our case, creating logical clauses for protein and location, and little of the clause is relevant to the relation *between* these two entities. We believe that using a named-entity classifier to identify promising pieces of our relation first could both reduce the number of examples as well as produce high quality clauses due to their direct focus on the relation. Blaschke et al. [4, 3] and Rindflesh et al. [25] have found success in biomedical information extraction using domain expert rules, and Temkin and Gilder [31] use handcrafted context-free grammars to similar ends. Another step in this direction is taking these clauses from a domain expert and learning to revise their advice, similar to work by Eliassi-Rad and Shavlik [13].

Finally, there are many more datasets in Information Extraction where we are planning to test our method for comparison, namely the genetic disorder and protein interaction from Ray and Craven [23] and a protein interaction dataset

from Brunescu et al. [7]. Other datasets outside of IE where we believe Gleaner will be useful include the nuclear smuggling dataset from Tang et al. [30], the social network dataset from Taskar et al. [2], and the CiteSeer citation dataset from Popescul et al. [21]

8 Acknowledgements

Our dataset can be found at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/datasets/IE-protein-location

This work was supported by National Library of Medicine (NLM) Grant 5T15 LM007359-02, NLM Grant 1R01 LM07050-01, DARPA EELD Grant F30602-01-2-0571, and United States Air Force Grant F30602-01-2-0571. We would like to thank Ines Dutra and Vitor Santos Costa for their help with Yap, the UW Condor Group for Condor assistance, Soumya Ray and Marios Skounakis for their help with labeling the data, and David Page for his help with Aleph, as well as the anonymous reviewers for their informative comments.

References

1. S. Aitken. Learning Information Extraction Rules: An Inductive Logic Programming Approach. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence*, Amsterdam, 2002.
2. M.-F. W. Ben Taskar, Pieter Abbeel and D. Koller. Label and Link Prediction in Relational Data. In *IJCAI Workshop on Learning Statistical Models from Relational Data*, 2003.
3. C. Blaschke, L. Hirschman, and A. Valencia. Information Extraction in Molecular Biology. *Briefings in Bioinformatics*, 3(2):154–165, 2002.
4. C. Blaschke and A. Valencia. Can Bibliographic Pointers for Known Biological Data be Found Automatically? Protein Interactions as a Case Study. *Comparative and Functional Genomics*, 2:196–206, 2001.
5. A. Bradley. The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.
6. L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
7. R. Bunescu, R. Ge, R. Kate, E. Marcotte, R. Mooney, A. Ramani, and Y. Wong. Comparative Experiments on Learning Information Extractors for Proteins and their Interactions. *Journal of Artificial Intelligence in Medicine*, 2004.
8. M. Califf and R. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.
9. C. Cortes and M. Mohri. AUC Optimization vs. Error Rate Minimization. In *Neural Information Processing Systems NIPS2003*, 2003.
10. M. Craven and S. Slattery. Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, 43(1/2):97–119, 2001.
11. I. de Castro Dutra, D. Page, V. S. Costa, and J. Shavlik. An Empirical Evaluation of Bagging in Inductive Logic Programming. In *Twelfth International Conference on Inductive Logic Programming*, pages 48–65, Sydney, Australia, 2002.

12. T. Dietterich. Machine-Learning Research: Four Current Directions. *The AI Magazine*, 18(4):97–136, 1998.
13. T. Eliassi-Rad and J. Shavlik. A Theory-Refinement Approach to Information Extraction. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.
14. D. Freitag and N. Kushmerick. Boosted Wrapper Induction. In *AAAI/IAAI*, pages 577–583, 2000.
15. S. Hoche and S. Wrobel. Relational Learning Using Constrained Confidence-Rated Boosting. In *11th International Conference on Inductive Logic Programming*, Strasbourg, France, 2001.
16. Z. Hu. Guidelines for Protein Name Tagging. Technical report, Georgetown University, 2003.
17. D. Kauchak, J. Smarr, and C. Elkan. Sources of Success for Boosted Wrapper Induction. *Journal of Machine Learning Research*, 5:499–527, May 2004.
18. N. Lavrac, F. Zelezny, and P. Flach. RSD: Relational Subgroup Discovery through First-order Feature Construction. In *Proceedings of the 12th International Conference on Inductive Logic Programming (ILP'02)*, Sydney, Australia, 2002.
19. C. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
20. R. Michalski and J. Larson. Inductive Inference of VL Decision Rules. In *Proceedings of the Workshop in Pattern-Directed Inference Systems*, May 1977.
21. A. Popescul, L. Ungar, S. Lawrence, and D. Pennock. Statistical Relational Learning for Document Mining. In *IEEE International Conference on Data Mining, ICDM-2003*, 2003.
22. M. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.
23. S. Ray and M. Craven. Representing Sentence Structure in Hidden Markov Models for Information Extraction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001)*, 2001.
24. E. Riloff. The Sundance Sentence Analyzer. *http://www.cs.utah.edu/projects/nlp/*, 1998.
25. T. Rindflesch, T. Tanabe, L. Weinstein, and J. Hunter. Edgar: Extraction of drugs, genes and relations from the biomedical literature. In *Proceedings of the Pacific Symposium on Biocomputing.*, 2000.
26. H. Shatkay and R. Feldman. Mining the Biomedical Literature in the Genomic Era: An Overview. *Journal of Computational Biology*, 10(6):821–55, 2003.
27. A. Srinivasan. The Aleph Manual Version 4. *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*, 2003.
28. A. Srinivasan and R. King. Feature Construction with Inductive Logic Programming: A Study of Quantitative Predictions of Biological Activity Aided by Structural Attributes. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 352–367. Stockholm University, Royal Institute of Technology, 1996.
29. A. Srinivasan, S. Muggleton, M. Sternberg, and R. King. Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.
30. L. Tang, R. Mooney, and P. Melville. Scaling up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism. In *KDD Workshop on Multi-Relational Data Mining*, 2003.
31. J. Temkin and M. Gilder. Extraction of Protein Interaction Information From Unstructured Text Using a Context-Free Grammar. *Bioinformatics*, 19(16):2046–2053, 2003.

A Monte Carlo Study of Randomised Restarted Search in ILP

Filip Železný[1], Ashwin Srinivasan[2], David Page[3]

[1] Dept. of Cybernetics
School of Electrical Engineering
Czech Institute of Technology (ČVUT) in Prague
Karlovo Nám. 13, 121 35 Prague, Czech Republic
zelezny@fel.cvut.cz
[2] IBM India Research Laboratory
Block 1, Indian Institute of Technology
New Delhi 110 016, India
ashwin.srinivasan@in.ibm.com
[3] Dept. of Biostatistics and Medical Informatics and Dept. of Computer Science
University of Wisconsin
1300 University Ave., Rm 5795 Medical Sciences
Madison, WI 53706, USA
page@biostat.wisc.edu

Abstract. Recent statistical performance surveys of search algorithms in difficult combinatorial problems have demonstrated the benefits of randomising and restarting the search procedure. Specifically, it has been found that if the search cost distribution (SCD) of the non-restarted randomised search exhibits a slower-than-exponential decay (that is, a "heavy tail"), restarts can reduce the search cost expectation. Recently, this heavy tail phenomenon was observed in the SCD's of benchmark ILP problems. Following on this work, we report on an empirical study of randomised restarted search in ILP. Our experiments, conducted over a cluster of a few hundred computers, provide an extensive statistical performance sample of five search algorithms operating on two principally different ILP problems (artificially generated graph data and the well-known "mutagenesis" problem). The sample allows us to (1) estimate the conditional expected value of the search cost (measured by the total number of clauses explored) given the minimum clause score required and a "cutoff" value (the number of clauses examined before the search is restarted); and (2) compare the performance of randomised restarted search strategies to a deterministic non-restarted search. Our findings indicate that the cutoff value is significantly more important than the choice of (a) the specific refinement strategy; (b) the starting element of the search; and (c) the specific data domain. We find that the optimal value for the cutoff parameter remains roughly stable across variations of these three factors and that the mean search cost using this value in a randomised restarted search is up to three orders of magnitude (i.e. 1000 times) lower than that obtained with a deterministic non-restarted search.

1 Introduction

Computer programs now collectively termed "predictive Inductive Logic Programming" (predictive ILP) systems use domain-specific background information and pre-classified sample data to construct a set of first-order rules for predicting the classification labels of new data. Despite considerable diversity in the applications of ILP, (see [3] for an overview) successful implementations have been relatively uniform, namely, engines that repeatedly examine sets of candidate rules to find the "best" ones. Any one step of this sequence is an enumerative search—usually some approximation to the optimal branch-and-bound algorithm—through a space of possible rules. This choice of search method can critically affect the performance of an ILP system on non-trivial problems. Enumerative search methods, despite their attractive simplicity, are not *robust* in the sense of achieving a balance between efficiency and efficacy across different problems [4]. For many practical problems that engender very large spaces of discrete elements, enumerative search, however clever, becomes intractable and we are forced to take seriously Trefethen's Maxim No. 30 [1]: "If the state space is huge, the only reasonable way to explore it is at random."

Recently, research into the development of efficient automatic model-checkers has led to the development of novel randomised search methods that abandon optimality in favour of "good" solutions. Prominent examples are the GSAT and WalkSat methods checking the satisfiability of propositional formulae [9], as randomised alternatives to the (enumerative) Davis-Putnam solver. In conjunction with this, there is now a vigorous line of research that investigates properties of large search spaces corresponding to difficult combinatorial problems [5]. Some intriguing properties have been identified, such as the high irregularity of the search spaces and "heavy-tailedness" of the cost distributions of search algorithms used. Such properties manifest themselves in a large collection of real-world problems and have been the inspiration for the design of randomised restarted search procedures. The basic idea of these procedures is simple: if each search trial has a small, but fixed probability of finding a good clause, then the probability of finding a good clause in a sequence of such trials can be made quite high very rapidly. Put differently, the SCD from the sequence has an exponential decay.

Previously, the heavy-tailed character of search cost distributions was reported in the context of the first-order rule search conducted in ILP [11]. There, a simple adaptation of a method known as Randomised Rapid Restarts [6] was shown to result in a considerable reduction of clause search cost. Here, we extend that investigation as follows:

1. We adapt a family of randomised restarted search strategies into an ILP system and present all of them as instantiations of a general algorithm.
2. We design and conduct an extensive Monte Carlo study that allows us to model the statistical relationships between the search cost, the score of the best clause and the number of clauses explored in each restart (called the "cutoff" value in the search algorithm).

Our experiments are conducted with data drawn from two domains: artificially generated, noise-free, graph problems, in which "target" theories can be modelled by a single, long clause (up to 10 literals in the body of the clause); and the well-known mutagenesis problem, which is typically modelled by multiple, relatively short clauses (typically up to 5 body literals). Although the natures of the problems are quite different to each other, the main statistical findings relate equally to both the domains.

The paper is organised as follows. In the next section we describe the clause search strategies considered and the performance metric used to evaluate the strategies. Details of the Monte Carlo study of these strategies and the dependence of their performance on some important parameters is in Section 3, where we also discuss our results and formulate questions requiring further investigation. Section 4 concludes the paper.

2 Search

We are principally concerned with performing a search in the clause subsumption lattice bounded at one end by a finite most specific ("bottom") clause derived using definitions in the background knowledge, a depth-bounded mode language, and a single positive example (the "saturant": see [8] for more details on the construction of this clause). For simplicity, we will assume the specification of the depth-bounded mode language to be part of the background knowledge.

2.1 Strategies

The five search strategies that we investigate in this paper are: (1) A deterministic general-to-specific search (DTD); (2) A randomised general-to-specific search (RTD); (3) A rapid random restart search (RRR); (4) A randomised search using the GSAT algorithm (GSAT); and (5) A randomised search using the WalkSAT algorithm (WSAT). All five strategies can be viewed as variations of a general search procedure shown in Fig. 1. Differences between the individual strategies arise from the implementation of the commands in bold-face (summarised in Table 1). All strategies include restarts (if γ is a finite value). Restarting DTD clearly results in simply repeating the search.

As further clarification of the entries in Table 1, we note the following:

Saturant selection. A deterministic implementation ('D') of the first **Select** command (Step 3 in Fig. 1) results in the first positive example in the presented example sequence is chosen as the saturant. A randomised implementation ('R') results all examples having a uniform probability of selection.

Start clause selection. A deterministic implementation ('D') of the second **Select** command (Step 4), results in the search commencing with the the most general definite clause allowable. A randomised implementation ('R') results in a clause selected with uniform probability from the set of allowable clauses (see [11] for more details on how this is achieved).

search$(B, H, E, s^{suf}, c^{all}, \gamma)$: Given background knowledge B; a set of clauses H; a training sequence $E = E^+, E^-$ (i.e. positive and negative examples); a sufficient clause score s^{suf} $(-\infty \leq s^{suf} \leq \infty)$; the maximum number of clauses the algorithm can evaluate c^{all}, $(0 < c^{all} < \infty)$; and the maximum number of clauses evaluated on any single retart or the 'cutoff' value γ $(0 < \gamma \leq \infty)$, returns a clause D such that $B \cup H \cup \{D\}$ entails at least one element e of E^+. If fewer than c^{all} clauses are evaluated in the search, then the score of D is at least s^{suf}.

1. $S := -\infty$; $C := 0$; $N := 0$
2. repeat
3. **Select** e^{sat} from E^+
4. **Select** D_0 such that $D_0 \succeq_\theta \bot(e^{sat}, B)$
5. $Active = \emptyset$; $Ref = \{D_0\}$
6. repeat
7. $S^* = \max_{D_i \in Ref} \underline{eval}_{B,H}(D_i)$; $D^* := \arg\max_{D_i \in Ref} \underline{eval}_{B,H}(D_i)$
8. if $S^* > S$ then $S := S^*$; $D := D^*$
9. $N := N + |Ref|$
10. $Active :=$ **UpdateActiveList**$(Active, Ref)$
11. $Prune :=$ **Prune**$(Active, S^*)$
12. $Active := Active \setminus Prune$
13. **Select** D^{curr} from $Active$; $Active := Active \setminus D^{curr}$
14. $Ref :=$ **Refine**$_{B,H,(\gamma-N)}(D^{curr})$
15. until $S \geq s^{suf}$ or $C + N \geq c^{all}$ or $N = \gamma$
16. $C := C + N$; $N := 0$
17. until $S \geq s^{suf}$ or $C \geq c^{all}$
18. if $S = -\infty$ then return e^{sat} else return D^*.

Fig. 1. A general skeleton of a search procedure—possibly randomised and/or restarted—in the clause subsumption lattice bounded by the clause $\bot(e^{sat}, B)$. This clause is derived using the saturant e^{sat} and the background knowledge B. In Step 4, $\succeq_\theta$ denotes Plotkin's (theta) subsumption between a pair of Horn clauses. Individual strategies considered in this paper are obtained by different implementations of the bold-typed commands. Clauses are scored by a finite evaluation function $\underline{eval}$. Although in the formal notation in Step 7 the function appears twice, it is assumed that the 'max' and 'arg max' operators are computed simultaneously. In Step 11 **Prune** returns all elements of *Active* that cannot possibly be refined to have a better score than S^*. If the number of refinements of the current clause is greater than $(\gamma - N)$, **Refine** returns only the first $(\gamma - N)$ computed refinements, to guarantee that no more than γ clauses are evaluated between restarts. The search is terminated when score s^{suf} is reached or c^{all} clauses have been evaluated, and restarted (from Step 3) when γ clauses have been evaluated since the last restart. If all **Select** commands are deterministic then restarting (setting $\gamma < c^{all}$) results in mere repetitions of the identical search.

Strategy → ↓ *Step*	DTD	RTD	RRR	GSAT	WSAT
Saturant selection (**Select** in Step 3)	D	R	R	R	R
Start clause selection (**Select** in Step 4)	D	D	R	R	R
Update active list (**UpdateActiveList** in Step 10)	C	C	C	G	G
Next clause selection (**Select** in Step 13)	D	R	D	D	R
Pruning (**Prune** in Step 11)	Y	Y	N	N	N
Refinement (**Refine** in Step 14)	U	U	B	B	B

Table 1. Implementation differences amongst the different search strategies. The entries are as follows: 'D' stands for 'deterministic', 'R' for 'randomised', 'G' for greedy, 'C' for complete, 'Y' to denote that pruning occurs, 'N' that pruning does not occur, 'U' for uni-directional refinement (specialisation only) and 'B' for to bi-directional refinement (specialisation and generalisation). See text for more details on these entries.

Update active list. A greedy implemention ('G') of the **UpdateActiveList** function (Step 10) results in the active list containing only the newly explored nodes (elements of the Ref). A complete implementation ('C') results in *Active* containing all elements (including elements of Ref).

Next clause selection. A deterministic implementation ('D') of the last **Select** command (Step 13) results in the clause with the highest score being chosen from the *Active* list (with ties being decided by the appearance order of clauses). A randomised implementation ('R') results in a random choice governed by the following prescription:

- With probability 0.5, select the clause with the highest score in the *Active* list.
- Otherwise, select a random clause in the *Active* list with probability proportional to its score.

Pruning. 'Y' denotes that pruning is performed, which results in a possibly non-empty set being returned by the **Prune** command (Step 11). A 'N' implementation means that an empty set is returned.

Refinement. The 'U' implementation of **Refine** command (Step 14) results in refinements that are guaranteed to be specialisations of the clause being refined. The 'B' implementation produces the (most general) specializations and (most specific) generalizations of the refined clause.

2.2 Evaluation

Informally, given some clause evaluation function, for each search strategy we ask:

How many clauses must be searched to achieve a desired clause score?

Here, we treat the number of clauses searched as representative of the 'search cost' and quantify this cost-score trade-off by the the expected value of the

smallest cost needed to achieve or exceed a desired score s^{suf} [1], given an upper bound γ on the clauses searched on any single restart. Thus, for each strategy we wish to estimate:

$$cost(s^{suf}) \equiv E[C|s^{suf}, \gamma] \tag{1}$$

The following points are evident, but worth restating:

1. Let us assume that strategy St_1 is found to achieve, on average, a desired score s^{suf} significantly faster than strategy St_2. Strictly speaking, even if the clauses added successively to a constructed theory do not reference each other, we cannot conclude that a set-covering algorithm employing St_1 will be more efficient than that using St_2. This is because in the cover algorithm, the individual clause search procedures are not statistically independent events (since one influences the following by removing a subset of the positive examples).[2]
2. We are only concerned here with the search cost in finding a clause with a given score on the training set. This does not, of course, translate to any statement about the performance of the clause found on new (test) data. It is certainly interesting and feasible to also investigate whether and how the generalization rate is statistically dependent on the procedure used to arrive at an acceptable clause, given a required score. This is, however, outside the scope of this study.
3. A search cost of immediate interest is the processor time occupied by a strategy. By adopting instead to measure the number of clauses searched, we are unable to quantify precisely the exact time taken by each strategy. Besides the obvious hardware dependence, research elsewhere [2] has shown that the cost of evaluating a clause can vary significantly depending on the nature of the problem addressed and formulation of the background knowledge. In this study we are concerned with obtaining some domain-independent insight into the five strategies.
4. Our evaluation of the strategies may have well been based on a different question, namely:

   What clause score is achieved given an allocated search cost?

   Here we would consider the expected score given an allocated cost c^{all}, that is $score(c^{all}) \equiv E[S|c^{all}, \gamma]$. Although the sample resulting from the Monte Carlo study can be used to evaluate the strategies in this way, space requirements confine this study to the former question, which we believe is of more immediate interest to practitioners of ILP.

[1] At any stage of the search, the score value maintains the highest clause evaluation so far obtained in the search. In other words, within a particular search execution, the score value is a non-decreasing function of the cost (i.e. the number of clauses searched).

[2] The conclusion would however be correct for many other ruleset induction algorithms where the events are independent, such as CN2-like unordered rulesets, various other voting rulesets etc.

3 Empirical Evaluation

3.1 Materials

Data Experiments were conducted using two ILP benchmarks. The first data set describes a set of 5,000 directed graphs (containing in total 16,000 edges). Every node in a graph is coloured to red or black. Each graph is labelled positive if and only if it contains a specific (coloured) subgraph which can be represented by a predefined clause of 10 body literals. Although this clause can be viewed as the target theory, there exist other clauses (subgraphs) in the search lattice that precisely separate the positive examples from the negatives. The second problem – mutagenesis prediction – has been discussed extensively in ILP literature and we use here one of the datasets described in our previous publication, namely the data pertaining to 188 "regression-friendly" compounds [10]. Table 2 describes the principal differences between the two data sets. The graph data set is available on request to the first author and the software for its generation can be obtained from the third author. The mutagenesis dataset can be obtained via anonymous ftp to `ftp.comlab.ox.ac.uk` in the directories `pub/Packages/ILP/Datasets/mutagenesis/aleph`.

Property	**Graphs**	**Mutagenesis**
Origin	Artificially generated	Real-life
Noise	No	Yes
'Target' theory	Yes	No
'Good' theory	One long clause (10 lits)	Several short clauses (up to 5 body lits)
# pos/neg examples	20/20	125/63

Table 2. Differences between the experimental data sets.

Algorithm and Machines All experiments use the ILP program Aleph. Aleph is available at: `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.pl`. Additional code implemented to Aleph for purposes of the empirical data collection can be obtained from the first author. The computation was distributed over the Condor computer cluster at the University of Wisconsin, Madison. All subsequent statistical analysis of the collected data was done by means of the R statistical package. The R procedures developed for this purpose can be obtained from the first author.

3.2 Method

Recall that we are interested in estimating the conditional expected cost value $E[C|s^{suf}, \gamma]$ for each of the five strategies in Section 2.1. A straightforward way

to collect the required statistical sample needed to estimate the expected value for a given search strategy would thus be to run a number of instances of the algorithm in Fig. 1, each with a different setting of the sufficient score parameter s^{suf} and the restart cutoff parameter γ, each time recording the resulting value of C. This approach would however perform a lot of redundant computation. Instead we adopt the following method:

For each problem (5,000 graphs and 188 mutagenic chemicals)
  For each randomized strategy (RTD, RRR, GSAT, WSAT)
    1. $\gamma = \infty$, $c^{all} = c_{max}$ (some large value: see notes below), $s^{suf} = s_{max}$ (the maximum possible clause score: see notes below)
    2. for $i = 1$ to $\#Runs$
       (a) Call $search(B, \emptyset, E, s^{suf}, c^{all}, \gamma)$ (see Fig. 1).
       (b) Record the 'performance' vector $c_i = [c_i(0), \ldots, c_i(s_{max_i})]$ where $c_i(s)$ is the number of clauses evaluated before achieving (or exceeding) score s for the first time and s_{max_i} is the maximum score achieved on run i.
    3. Compute the expected cost from the performance vectors recorded.

The following details are relevant:

1. The method assumes a finite, integer-valued scoring function. In the experiments we evaluate the score of a clause D as $P(D) - N(D)$ where $P(D)$ and $N(D)$ are the numbers of positive and negative examples 'covered' by D. That is, given positive and negative examples E^+, E^- let $E_p \subseteq E^+$ s.t. $B \cup \{D\} \models E_p$ and $E_n \subseteq E^-$ s.t. for all $e_n \in E_n$, $B \cup \{D\} \cup \{e_n\} \models \Box$. Then $P(D) = |E_p|$ and $N(D) = |E_n|$. Rejecting all clauses for which $P(D) < N(D)$, the range of the score is $0 \ldots P$ where $P = |E^+|$. Thus in Step 1 $s_max = P$.
2. In these experiments c_{max} was set to 200,000 and $\#Runs$ was set to 6,000. Thus, the empirical sample after execution of Step 2 consists of 6,000 performance vectors. Each performance vector has at most $P = |E^+|$ elements (fewer elements are possible if score P was not achieved on a run).
3. Step 3 requires the computation of expected cost (i.e. the number of evaluated clauses) for any value of γ and s^{suf}. In Appendix A we describe how the sample of 6,000 performance vectors can be used to obtain an unbiased estimate this value.

The method above does not refer to the non-restarted strategy DTD. DTD is deterministic and thus a single run (and corresponding single performance vector) should be sufficient to describe its performance. However, unlike the other strategies, DTD does not select the saturated example randomly, but it selects the first positive example for saturation. To avoid artifacts arising from any particular example ordering, we obtain instead an average conditional cost. That is, we thus perform Step 2a above $P = |E^+|$ times, each time selecting a different saturant. This results in P performance vectors: Appendix A shows how these performance vectors can be used to obtain a biased (lower bound) estimate of the expected cost for any value of s^{suf}.

3.3 Results

Figures 2–5 show diagrammatically the estimated expected number of evaluated clauses ('expected cost value') as a function of the restart cutoff parameter γ. Each graph has 2 kinds of plots: horizontal lines, representing (a lower bound on) the cost of the deterministic strategy DTD; and non-constant lines representing a randomised strategy. Each line is tagged with a number, which represents the sufficient clause score s^{suf}. Thus, in Fig. 2, for the plot labelled "Mutagenesis: RTD", the horizontal line tagged "10" with a constant cost of 10,000 indicates that the expected number of clauses explored by DTD before finding a clause with score 10 is 10,000. The corresponding costs, for different values of the cutoff parameter γ for RTD is shown by the lowermost non-constant line (each entry is tagged by "10"). Figures associated to the graphs domain contain 20 such plots (corresponding to all possible clause scores: some plots may overlay), figures belonging to the mutagenesis domain contain 7 plots (corresponding to $s^{suf} = 10, 20, \ldots 70$).[3] In all the diagrams, the highest vertical (*cost*) coordinate should be interpreted as: "c_{max} or higher" as all points corresponding to an expected cost in the interval $[c_{max}, \infty]$ are plotted with the vertical coordinate c_{max}.

Broadly, there are remarkable similarities in the plots from different strategies for a given problem domain, as well as from the same strategy for different problem domains. The principal trends are these:

1. The setting of the cutoff parameter γ has a very strong impact on the expected cost. A choice close to its optimal value may reduce the expected cost over DTD up to a 1,000 times for mutagenesis with RRR, GSAT or WSAT[4]. With RTD, the reduction is even higher for very low s^{suf}.
2. $\gamma \approx 100$ is a 'good' choice for all the investigated strategies in both domains. For the graph problems, the value is close to optimal (in the considered range of γ) for all strategies and for all of them it leads to similar expected costs, with the exception of WSAT where the costs are significantly higher[5] In mutagenesis, $\gamma \approx 100$ is a local minimum for all strategies (for RTD it is even the optimum) when $s^{suf} > 30$.
3. $\gamma < 10$ appears to be uniformly a 'bad' choice for all randomised strategies on the graph problems.
4. For a given domain, expected costs of the different restarted strategies are roughly similar, especially in the vicinity of $\gamma = 100$ (once again, WSAT on the graphs problem is the exception).

[3] We do not plot lines for further scores $s^{suf} = 80, 90 \ldots P$ since these were not achieved by any of the strategies in any of the runs.

[4] In mutagenesis, the high expected costs of DTD are due to several positive examples, whose saturation leads to an unfavorable search space.

[5] A tentative explanation of this may lie in the fact that the explored clause score is a particularly good heuristic for node selection in the noise-free graph problems. In these circumstances the randomization inherent in WSAT may be having a detrimental effect on the mean performance of WSAT.

5. For both domains, other than very high values of s^{suf}, the costs of the restarted strategies are uniformly lower than that of the non-restarted strategy DTD for a wide range of γ ($100 \leq \gamma \leq 10,000$).

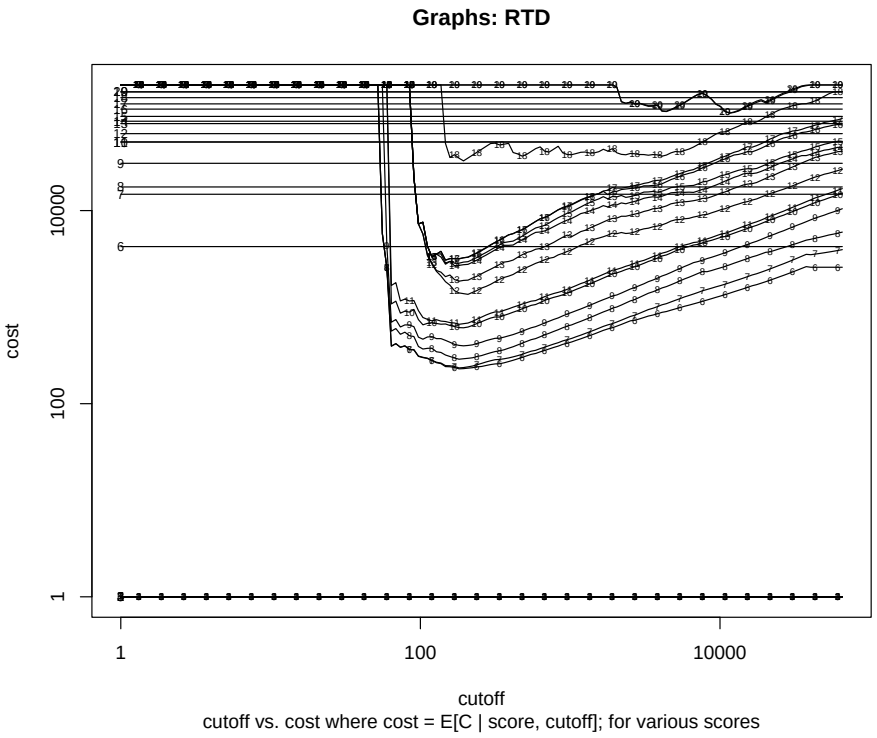
3.4 Discussion

For large intervals of the cutoff parameter γ, the restarted randomised search strategies (RTD, RRR, GSAT, WSAT) exhibit similar performance, outperforming the non-restarted deterministic clause search (DTD) in terms of the cost. However, two issues require further investigation before conclusions can be made concerning the ranking of the strategies. First, DTD (and RTD) always begins the search with the most general definite clause allowed by the language restriction. It is therefore biased towards evaluating shorter clauses than RRR, GSAT or WSAT which often means spending less total evaluation time for the same number of clauses scored. If processor time is assigned to the search cost, it is possible that both top-down strategies may improve their relative performance with respect to the restarted methods (of course, the empirical results suggest that the latter may evaluate far fewer clauses). Second, our measurement scheme does not assign any cost to the computation needed to construct a bottom clause for each restart. Clearly, the corresponding overhead time grows linearly with decreasing γ (there will be more restarts). As a result, when computation time is viewed as the search cost, the optimum value of γ is likely to shift to a higher value than that determined in our experiments, and the large performance superiority of the the restarted methods observed for small γ is likely to reduce.

It is encouraging that two apparently very different domains and all restarted strategies have yielded a similar range of 'good' values for the γ parameter. However, the plots, especially on in the graphs domain, highlight a further aspect: there is quite a sharp increase in costs for values of γ that are just below the optimum. This suggests that it would thus be useful to consider a restarted algorithm that is less dependent on the location of the optimal γ. A solution may lie in a cutoff value gradually (for example, geometrically) growing with each restart. This idea of *dynamic* restarts has been considered before [7] and may result in a more robust search algorithm.

4 Concluding Remarks

Search is at the heart of all modern Inductive Logic Programming systems, and most have thus far employed well-known deterministic search methods. In other fields confronted with very large search spaces, there is now substantial evidence that the use of randomised restarted strategies yield far superior results to deterministic ones (often making the difference between getting a good solution, or none at all). Unfavourable conditions for deterministic search observed in those fields—the heavy-tailed character of search cost distributions—have also been reported in the context of the search conducted by many ILP systems [11]. In



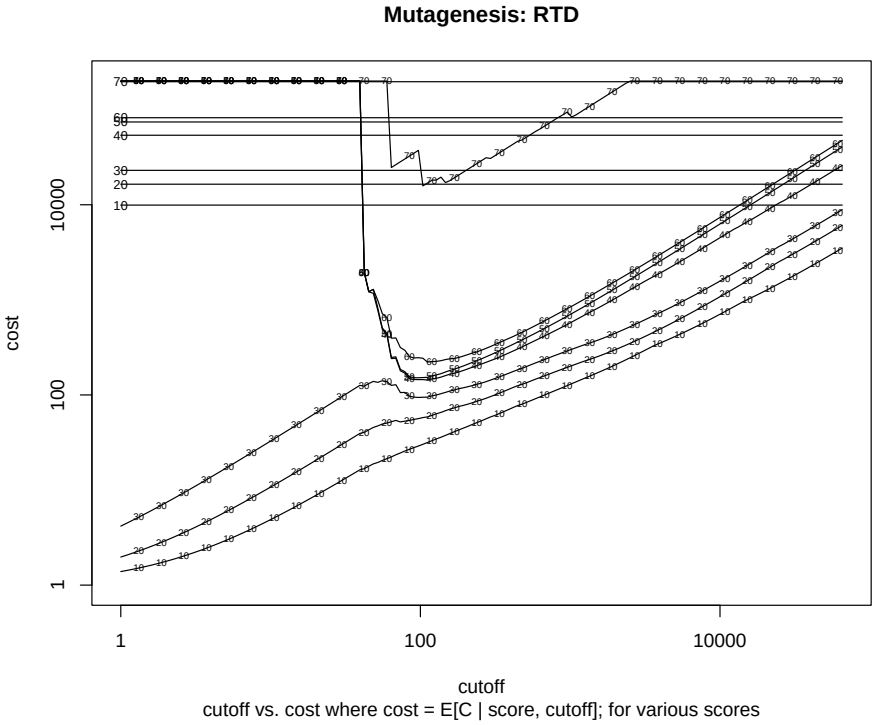


Fig. 2. Randomised Top-Down vs. Deterministic Top-Down (horizontal lines) search

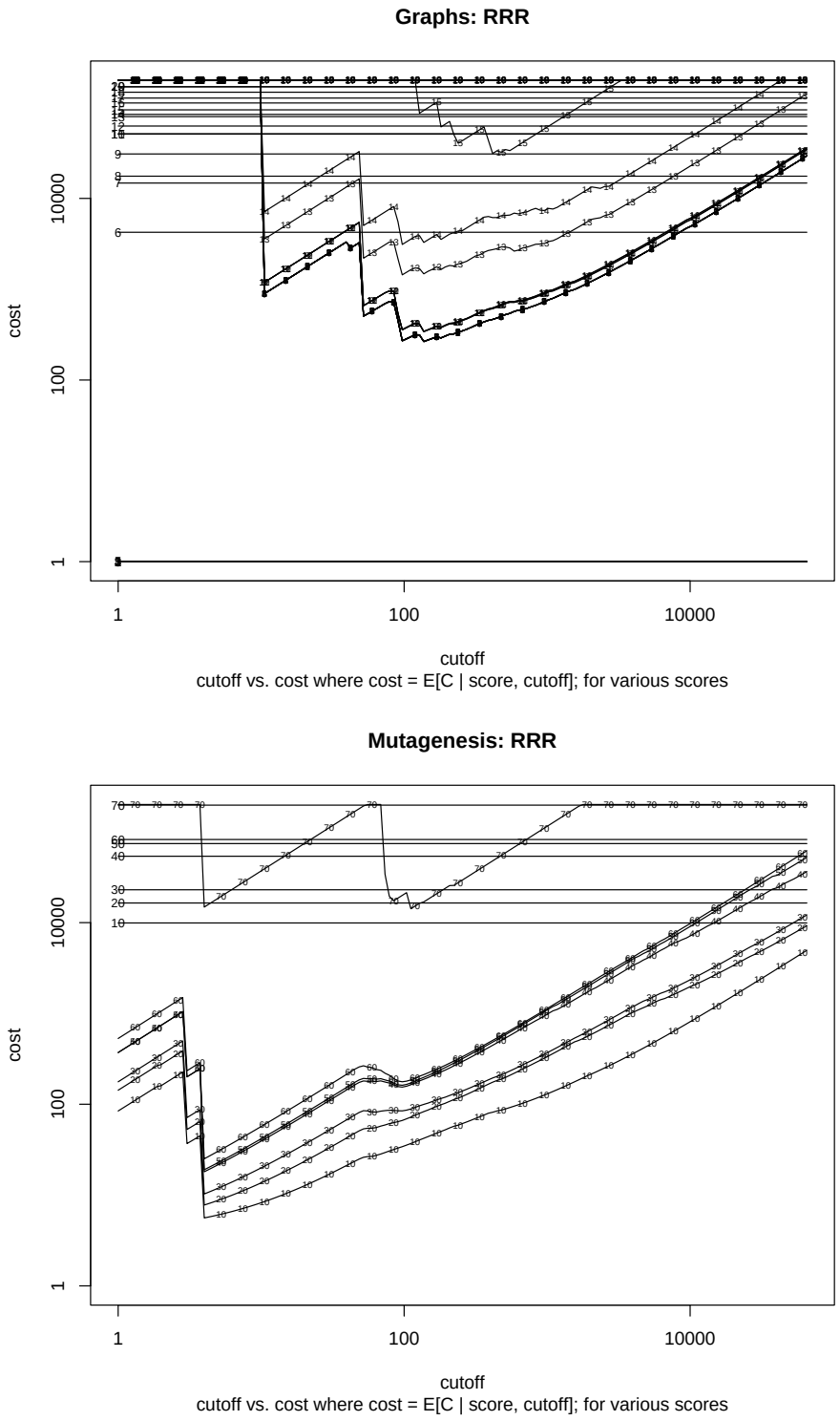


Fig. 3. 'Randomised Rapid Restarts' vs. Deterministic Top-Down (horizontal lines) search

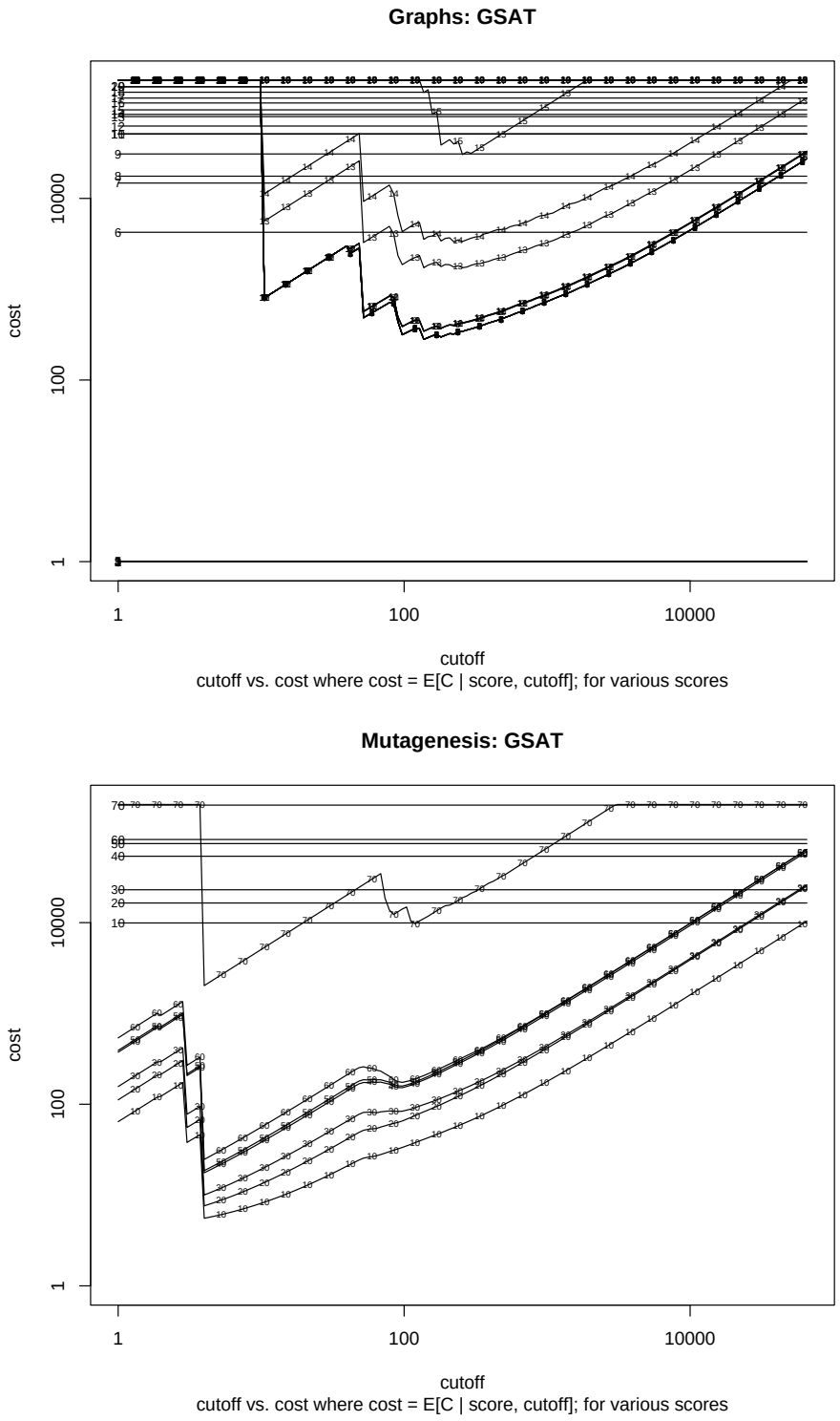


Fig. 4. GSAT vs. Deterministic Top-Down (horizontal lines) search

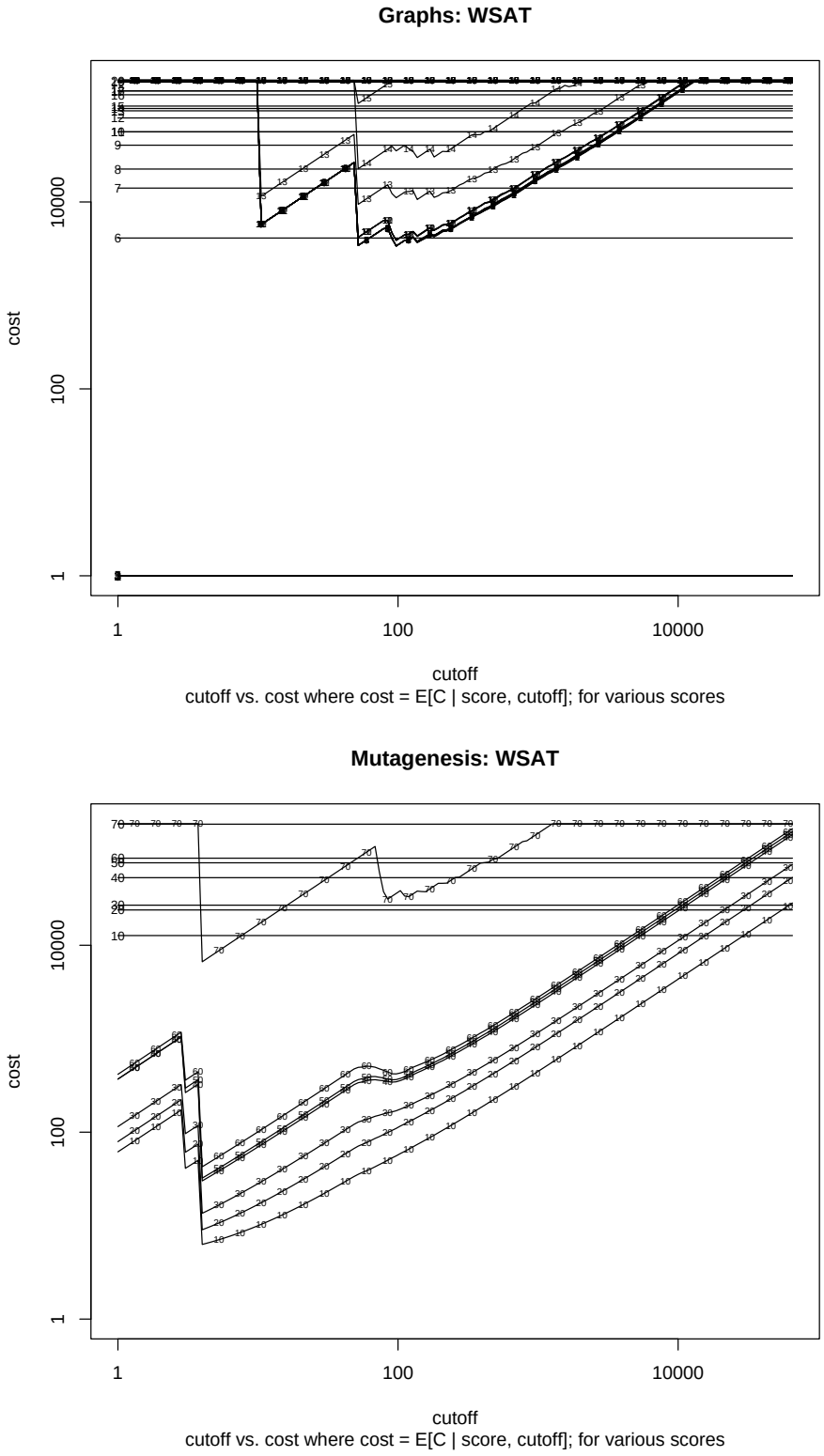


Fig. 5. WSAT vs. Deterministic Top-Down (horizontal lines) search

this paper, we have presented what appears to be the first systematic study of a number of randomised restarted search strategies for ILP. Specifically, we have adopted a Monte Carlo method to estimate the search cost—measured by the number of clauses explored before a 'good' clause is found—of these strategies on two quite different ILP problems. The result is encouraging: in each domain, for a wide range of values for a parameter γ controlling the number of restarts, randomised restarted methods have a lower search cost than a deterministic general-to-specific search.

The performance sample generated has also provided some useful insights into the randomised techniques. First, it appears that there may a 'good' value for γ that works adequately for across many domains. Second, although they differ in the choice of the first element of the search and the refinement strategy employed, all randomised methods appear to perform similarly. Third, there may be some value in exploring a randomised restarted search strategy with a dynamically growing value of γ.

While accepting all the usual caveats that accompany an empirical study such as this, we believe the results here to be sufficiently encouraging for researchers to explore further the use of randomised restarted search methods in ILP, especially with other data domains and scoring functions.

A Calculating Expected Cost from Performance Vectors

We describe here a technique for estimating the conditional expected cost value $E[C|s^{suf}, \gamma]$ for each of the five strategies in Section 2.1. We exploit the fact that the expected cost for a strategy with arbitrarily set s^{suf} and γ parameters can be estimated from a sample of executions of the algorithm in Fig. 1 where $s^{suf} = P$ (where P is the maximum possible score) and $\gamma = \infty$. Since we require all trials terminate in a finite time, we let c^{all} equal to some large finite value c_{max} (for the experiments in the paper $c_{max} = 200{,}000$) for each of the random trials. As we shall see below, setting a finite c^{all} will bias the estimates for non-restarted strategies, but will still allow us to obtain unbiased estimates for restarted strategies for all values of $\gamma < c^{all}$.

Recall that executing the experimental method described in Section 3.2 results, for each strategy and problem, in a set of 'performance' vectors $c_i = [c_i(0), \ldots, c_i(s_{max_i})]$ where $1 \leq i \leq 6000$ for the randomised strategies RTD, RRR, GSAT and WSAT; and $1 \leq i \leq P = |E^+|$ for the deterministic strategy DTD. With each c_i $c_i(s)$ is the number of clauses evaluated before achieving (or exceeding) score s for the first time and s_{max_i} is the maximum score achieved on run i

For DTD, $E[C|s^{suf}, \gamma = \infty]$ is obtained by simply averaging the $c_i(s^{suf})$ over all i's. However, it is possible that in some of the trials i, the maximum score P is not achieved after evaluating c_{max} clauses. In such cases, there exist values $s^{suf} \leq P$ such that $c_i(s^{suf})$ is not defined. Here we set $c_i(s^{suf}) \equiv c_{max}+1$. Thus, the cost we associate to non-restarted strategies will represent *lower bounds* of their expected cost.

For the restarted searches RTD, RRR, GSAT and WSAT, let the sequence of steps 4–14 in Fig. 1 be called a *try*. The probability that s^{suf} is achieved in the t-th try (and not in tries $1 \ldots t-1$), given the cutoff value γ, is

$$F(\gamma|s^{suf}) \left(1 - F_s(\gamma|s^{suf})\right)^{t-1} \tag{2}$$

where the conditional cumulative distribution $F(x|s) = P(C \leq x|s)$ represents the probability of achieving or exceeding the score s having evaluated x of fewer clauses. It is estimated for a given score s from the empirical data as the fraction

$$F(x|s) \approx \frac{|\{c_i|c_i(s) \leq x\}|}{|\{c_i\}|} \tag{3}$$

Note that the consequence of s not being achieved in a particular run i is simply that the condition $c_i(s) \leq x$ does not hold. That is, run i is counted as a realization of the random trial with an 'unsuccessful' outcome. Thus to estimate $F(x|s)$ we do not need to assign a value to $c_i(s)$ in such a case (as was done above for DTD) and the the estimate remains unbiased.
The expected number of tries initiated before achieving s^{suf} is

$$E[T|s^{suf}, \gamma] = F(\gamma|s^{suf}) \sum_{t=1}^{\infty} t \left(1 - F(\gamma|s^{suf})\right)^{t-1} \tag{4}$$

It equals 1 for $F(\gamma|s^{suf}) = 1$ and for $F(\gamma|s^{suf}) = 0$ we set $E[T|s^{suf}, \gamma] = \infty$. If $0 < F(\gamma|s^{suf}) < 1$, it can be shown that Expression 4 converges to

$$E[T|s^{suf}, \gamma] = \frac{1}{F(\gamma|s^{suf})} \tag{5}$$

If we simply assumed that the algorithm evaluates exactly γ clauses in each try including the last, then the expected number of evaluated clauses would be

$$\bar{E}[C|s^{suf}, \gamma] = \gamma E[R|s^{suf}, \gamma] = \frac{\gamma}{F(\gamma|s^{suf})} \tag{6}$$

However, $\bar{E}[C|s^{suf}, \gamma]$ is imprecise because the algorithm may achieve s^{suf} evaluating fewer than γ clauses in the last try. The expected total number of clauses evaluated in all but the last try is

$$\gamma E[T-1|s^{suf}, \gamma] = \gamma \left(\frac{1}{F(\gamma|s^{suf})} - 1\right) \tag{7}$$

Due to the linearity of the expectation operator, we can determine the correct total expected cost by adding to the above value the expected number of clauses evaluated in the last try. For this purpose, consider the family of conditional probability distributions

$$D_t(n) = P(N = n|(t-1) * \gamma < C \leq t\gamma, s^{suf}, \gamma) \tag{8}$$

For $t = 1, 2, \ldots$, each D_t describes the probability distribution of the number of evaluated clauses in the t-th try under the specified parameters s^{suf}, γ, and *given* that the t-th try is the last in the search, ie. an acceptable clause is found therein. Since individual tries are mutually independent, the distributions D_t are identical for all t, that is, for an arbitrary t it holds $D_t(n) = D_1(n)$. Because in the first try it holds[6] that $N = C$, we can write

$$D_1(n) = P(C = n | C \leq \gamma, s^{suf}) \tag{9}$$

We did not include γ in the conditional part because its value does not affect the probability of the event $C = n$ given that $C \leq \gamma$, ie. given that no restart occurs. Applying basic probability algebra,

$$D_1(n) = \frac{P(C = n, C \leq \gamma | s^{suf})}{P(C \leq \gamma | s^{suf})} \tag{10}$$

If $n > \gamma$ then $D_1(n) = 0$. Otherwise, we can drop the $C \leq \gamma$ conjunct (implied by $C = n$) from the nominator expression:

$$D_1(n) = \frac{P(C = n | s^{suf})}{P(C \leq \gamma | s^{suf})} = \frac{F(n | s^{suf}) - F(n-1 | s^{suf})}{F(\gamma | s^{suf})} \tag{11}$$

Now we can calculate the expected number $E[N | (t-1) * \gamma < C \leq r\gamma, s^{suf}, \gamma]$ of clauses evaluated in the last try as

$$\sum_{n=1}^{\infty} n D_t(n) = \sum_{n=1}^{\gamma} n D_1(n) = \sum_{n=1}^{\gamma} n \frac{F(n | s^{suf}) - F(n-1 | s^{suf})}{F(\gamma | s^{suf})} \tag{12}$$

Summing up Eq. 7 with Eq. 12 we get the expected total number of evaluated clauses:

$$E[C | s^{suf}, \gamma] = \gamma \left(\frac{1}{F(\gamma | s^{suf})} - 1 \right) + \frac{\sum_{n=1}^{\gamma} n \left(F(n | s^{suf}) - F(n-1 | s^{suf}) \right)}{F(\gamma | s^{suf})} \tag{13}$$

Recall that the conditional distribution $F(.|.)$ used above can be estimated from the performance vectors as described by Eq. 3.

References

1. *http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/maxims.html*.
2. M. Botta, A. Giordana, L. Saitta, and M. Sebag. Relational learning as search in a critical region. *Journal of Machine Learning Research*, (4):431–463, 2003.
3. S. Dzeroski. Relational data mining applications: An overview. In *Relational Data Mining*, pages 339–364. Springer-Verlag, September 2001.

[6] Within the first try, the total number of evaluated clauses equals the number of clauses evaluated in the current try.

4. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
5. C. Gomes and B. Selman. On the fine structure of large search spaces. In *Proceedings the Eleventh International Conference on Tools with Artificial Intelligence ICTAI'99, Chicago, IL*, 1999.
6. C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.
7. H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies, proceedings of the eighteenth.
8. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
9. B. Selman, H. J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.
10. A. Srinivasan, S. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.
11. F. Železný, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. volume 2583, pages 333–345, 2003.

Establishing Identity Equivalence in Multi-Relational Domains

Jesse Davis, Inês Dutra, David Page, Vítor Santos Costa
Dep of Computer Science and Dep of Biostatistics and Medical Informatics
University of Wisconsin-Madison
{jdavis,dutra,page,vitor}@biostat.wisc.edu

Keywords: Information Extraction and Link Analysis, Knowledge Discovery and Dissemination

Abstract

Identity Equivalence or Alias Detection is an important topic in Intelligence Analysis. Often, terrorists will use multiple different identities to avoid detection. We apply machine learning to the task of determining Identity Equivalence. Two challenges exist in this domain. First, data can be spread across multiple tables. Second, we need to limit the number of false positives. We present a two step approach to combat these issues. First, we use Inductive Logic Programming to find a set of rules that are predictive of aliases. In the second step, we treat each learned rule as a random variable in a Bayesian Network. We use the Bayesian Network to assign a probability that two identities are aliases. We evaluate our technique on several data sets and find that layering Bayesian Network over the rules significantly increases the precision of our system.

1 Introduction

Determining *Identity Equivalence*, or Alias Detection, is a common problem in Intelligence Analysis. Two different identifiers are equivalent or *aliases* if both refer to the same object. One traditional example of aliasing centers around mistyped or variant author names in documents. For example, one might want to determine if a citation for V.S. Costa and one for Vítor S. Costa refer to the same author. In this situation one evaluates matches based on textual similarity. Furthermore, the central information comes from the surrounding text (Pasula et al. 2002). However, Intelligence Analysis involves more complex situations, and often syntactic similarity is either unavailable or inapplicable. Instead, aliases must be detected through object attributes and through interaction patterns.

The Intelligence Analysis domain offers two further challenges to this problem. First, information is commonly stored in relational database management systems (RDBMS) and involves multiple relational tables. The format of the data suggests using multi-relational datamining techniques such as Inductive Logic Programming (ILP). ILP systems learn *rules*, which can contain fields from different tables, in First Order Logic. Modern ILP systems can handle significant databases, containing millions of tuples.

A second challenge in Intelligence Analysis arises from *false positives*, or false alarms. In our task, a false positive corresponds to incorrectly hypothesizing that two names refer to the same individual. A false positive might have serious consequences, such as incorrectly adding individuals to a no-fly list. False positives will always cause valuable time to be wasted. False positives reduce trust in the system: if an expert system frequently gives spurious predictions, analysts will ignore its output. For all of these reasons, it is essential to limit the false positive rate. Unfortunately, intelligence analysis is particularly susceptible to false positives, as one is often trying to detect anomalies that occur rarely in the general population. For example, in a city with 1 million individuals, there are 499 billion possible alias pairs. In this case, even a false positive rate of only 0.001% will result in about 5 million false positives, or about five bad aliases per person.

We propose a two step methodology to address these challenges. First, we learn a set of rules that can achieve high recall, that is, they should be able to recognize most of the true aliases. Unfortunately, some of these rules may have poor precision, meaning that they falsely classify identity pairs as aliases. The second step addresses this problem. Instead of just considering each rule as an individual classifier, we treat each rule as a feature of a new classifier. We use machine learning methods to obtain a classifier that takes advantage of the characteristics of the individual rules. We use Bayesian Networks as our model, as they calculate the probability that a pair of identities are aliases.

We have evaluated our approach on synthetic datasets developed by Information Extraction & Transport, Inc. within the EAGLE Project (Schrag 2004). We were provided with artificial worlds, characterized by *individuals*, and relationships between these individuals. Our results show excellent performance for several of the datasets.

This paper is organized as follows. In Section 2 we discuss ILP applied to alias detection. In Section 3 we give a brief overview of Bayesian networks. In Section 4 we present and discuss our results. We compare our work with related work in Section 5. Finally, in Section 6 we provide a more in depth discussion of the datasets and our results.

2 ILP For Alias Detection

Inductive Logic Programming (ILP) is a framework for learning relational descriptions (Lavrac and Dzeroski 2001). Given sets of positive and negative examples and background knowledge, an ILP system learns a set of rules to discriminate between the positive and negative instances. ILP is appropriate for learning in multi-relational domains as the learned rules are not restricted to contain fields or attributes for a single table in a database.

We use Srinivasan's Aleph ILP System (Srinivasan 2001). Aleph uses the Progol algorithm (Muggleton 1995) to learn rules described as Prolog programs. Aleph induces rules in two steps. Initially, it selects an example and searches the databases for the facts known to be true about that specific example. The Progol algorithm is based on the insight that some of these facts should explain this example. If so, it should be possible to generalize those facts so that they would also explain the other examples. The algorithm thus generates generalized combinations of the facts, searching for the combinations with best performance.

One major advantage of using ILP is that it produces understandable results. We show a sample rule generated by Aleph:

alias(Id_1,Id_2) ←
    suspect(Id_2),
    suspect(Id_3),
    phonecall(Id_2,Id_3),
    phonecall(Id_3,Id_1).

The rule says that two individuals Id_1 and Id_2 may be aliases if **(i)** they both made phone calls to the same intermediate individual Id_3; and **(ii)** individuals Id_2 and Id_3 have the same attribute (suspect). The rule reflects that in this world model suspects are more likely to have aliases. Moreover, an individual and its aliases tend to talk to the same people.

The next rule uses different information:

alias(Id_1,Id_2) ←
    has_capability(Id_1,Cap),
    has_capability(Id_2,Cap),
    group_member(Id_1,G),
    group_member(Id_2,G),
    isa(G,nonthreatgroup).

Two individuals may be aliases because they have a common capability, and because they both belong to the same nonthreat group.

Clearly, these two rules are not precise as the patterns these rules represent could easily be applied to ordinary individuals. One observation is that we are only using the original database schema. An analyst might define views, or inferred relations, that highlight interesting properties of individuals. For instance, the first rule indicates that an individual and its aliases tend to communicate with the same people. We thus might want to compare sets of people an individual and its aliases talk to. In the spirit of aggregate construction for multi-relational learning (Knobbe et al. 2001; Neville et al. 2003; Perlich and Provost 2003), we have experimented with hand-crafting rules that use aggregates over properties commonly found in the ILP learned rules.

Even inventing new attributes, it is impossible to find a single rule that correctly identifies all aliases. In the next section, we discuss our approach for combining rules to form a better classifier.

3 Bayesian Networks

One of the drawbacks of applying ILP to this problem is that each database for a world is extremely large. The consequence is that it is intractable to use all the negative examples when learning the rules, which makes the final set of rules more susceptible to false positives. First, by sampling the negative examples, we have changed the proportion of the population that has aliases. Second, in ILP the final classifier traditionally consists of forming a disjunction over the learned clauses, resulting in a decision list. An unseen example is applied to each clause in succession until it matches one of them. If the example does not match any rule, then it receives the negative classification. Unfortunately, the disjunction of clauses maximizes the number of false positives. These issues suggest a simple approach where we represent each learned rule as an attribute in a classifier. We used Bayesian networks to combine the rules for two reasons. First, they allow us to set prior probabilities to reflect the true proportion of the population that has aliases. Second, each prediction has a probability attached to it. We can view the probability as a measure of confidence in the prediction. We experiment with several different Bayes net models for combining the rules. Naïve Bayes (Pompe and Kononenko 1995) is straightforward approach that is easy to understand and fast to train.

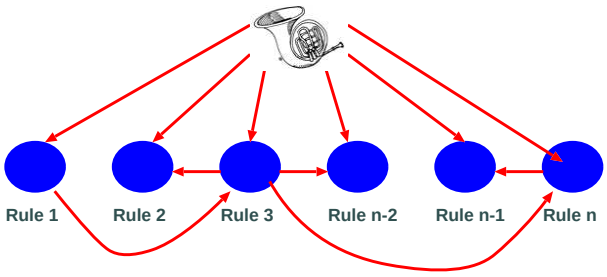


Figure 1: A TAN Bayes Net.

The major drawback with Naïve Bayes is that it assumes that the clauses are independent of each other given the class value. Often, we expect the learned rules to be strongly related. We used Tree Augmented Naïve Bayes networks (TAN) (Friedman et al. 1997), which allows for a slightly more expressive model. Figure 1 shows an example of a TAN network. Friedman, Geiger and Goldszmidt evaluated the algorithm for its viability on classification tasks. The TAN model, retains the basic structure of Naïve Bayes, but also permits each attribute to have at most one other parent, allowing the model to capture dependencies between attributes. To decide which arcs to include in the augmented network,

the algorithm makes a complete graph between all the non-class attributes, where the weight of each edge is given as the conditional mutual information between those two attributes. A maximum weight spanning tree is constructed over this graph, and the edges that appear in the spanning tree are added to the network. Geiger proved that the TAN model can be constructed in polynomial time with a guarantee that the model maximizes the Log Likelihood of the network structure given the dataset (Friedman et al. 1997).

We have also experimented with other structure learning approaches, such as the Sparse Candidate algorithm (Friedman et al. 1999), but did not obtain significant improvements, as discussed by Davis et al. (2004).

4 Experiments

This section presents our results and analysis of the performance of our system on EAGLE datasets (Schrag 2004). The datasets are generated by simulating an artificial world with large numbers of relationships between agents. The data focuses on *individuals* which have a set of attributes, such as the capability to perform some actions. Individuals may also obtain resources, which might be necessary to perform actions. Individuals belong to groups, and groups participate in a wide range of *events*. In our case, given that some individuals may be known through different identifiers (e.g., through two different phone numbers), we were interested in recognizing whether two identifiers refer to the same individual.

The EAGLE datasets have evolved toward more complex and realistic worlds. We evaluate our system for datasets generated by two versions of the simulator. The results from the first version of the simulator are indexed with numbers while the newer datasets are indexed by roman numerals. Datasets vary with size, both in the number of individuals and in the activity level of each individual. Datasets also differ on observability, the amount of information available as evidence; on corruption, the number of errors; and on clutter, the amount of irrelevant information. Each dataset includes pre-processed data, called *primary* data, with group information, and on *secondary* data. The primary data contains a number of presumed aliases, which may or may not be true.

Each experiment was performed in two rounds. In the first round, the *dry-run*, we received a number of datasets plus their corresponding ground truth. This allowed us to experiment with our system and validate our methodology. In the second round, the *wet-run*, we received datasets without ground truth and were asked to present a hypothesis. We had a limited amount of time to do so. Later, we received the ground truth so that we could perform our own evaluation.

We adopted the following methodology. Rule learning is quite expensive in these large datasets. Moreover, we have found that most rules are relevant across the datasets, as we believe they capture aspects common to each simulated world. Consequently, we only performed rule learning during the dry-run. We used Srinivasan's Aleph ILP system (Srinivasan 2001) running on the YAP Prolog system. Ground-truth was used for training examples (and not used otherwise). The best rules from all datasets were passed forward to the wet-run.

We present results on the wet-run data in this paper. For the first set of data we used the wet-run datasets plus group information derived by the Kojak system (Adibi et al. 2004) (we did not have access to this information for the second batch of data). Using the rules learned from the training data, we converted each of the evaluation datasets into a set of propositional feature vectors, where each rule appears as an attribute in the feature vector. Each rule served as a binary attribute, which received a value of one if the rule matched the example and a zero otherwise.

We first report results from an earlier version of the EAGLE simulator, where only a single alias was allowed per entity. For space reasons, we only show results for three out of six of the datasets. Results for the other three are similar. We used five fold cross validation in these experiments.

For each application we show precision versus recall curves for the three methods: Naïve Bayes, TAN and voting. We used our own software for Naïve Bayes and TAN.

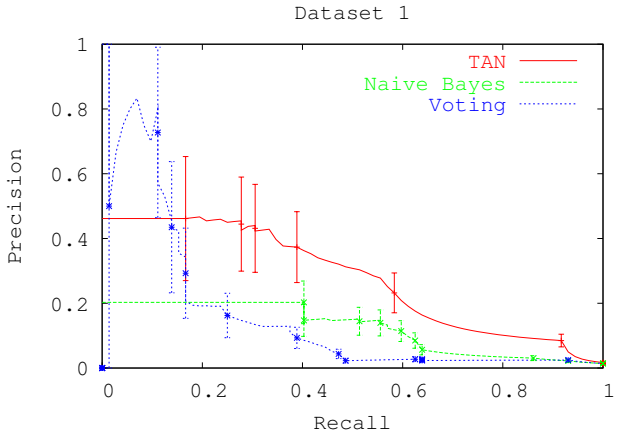


Figure 2: Precision Recall for Dataset 1

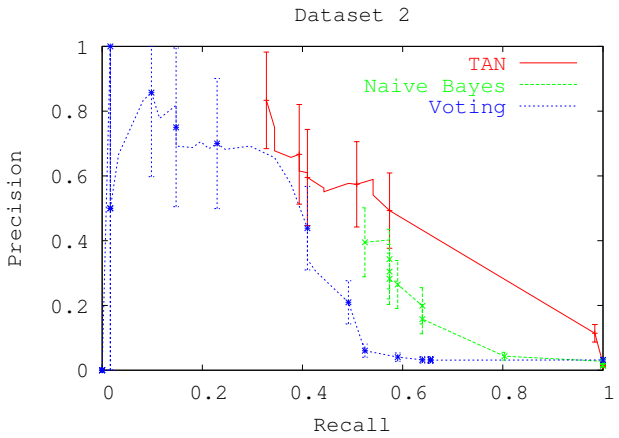


Figure 3: Precision Recall for Dataset 2

The Precision Recall curves for the different datasets are seen in Figures 2 through 4. We compare TAN and Naïve

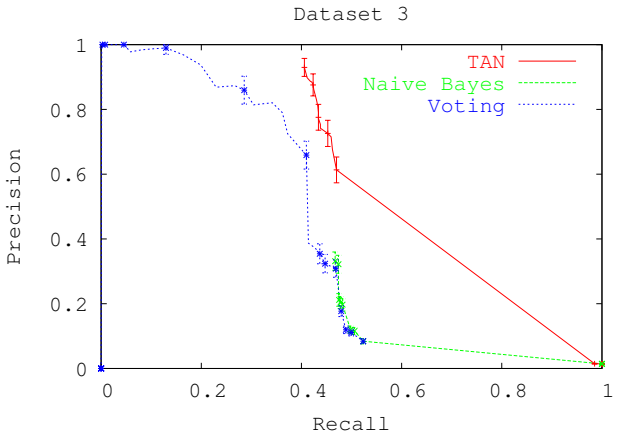


Figure 4: Precision Recall for Dataset 3

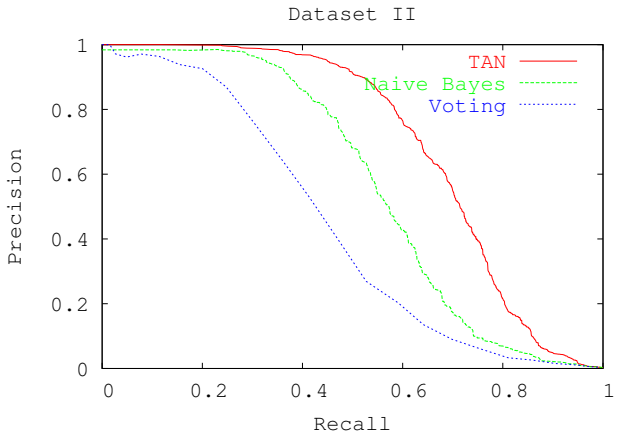


Figure 6: Precision Recall for Dataset II

Bayes with unweighted voting, an ensemble method (Davis et al. 2004). On each curve, we included 95% confidence intervals on the precision score for selected levels of recall. The curves were obtained by averaging the precision and recall values for fixed thresholds. We achieved the best results for datasets 2 and 3, and did the worst on dataset 1, where precision levels did not achieve 0.5.

The second set of results comes from a more recent version of the simulator. Dataset sizes were at least as large, or bigger than before. The new simulator supported social network attributes, which could be used for the aliasing task. The error levels were increased and each individual could have up to six aliases. We used the same methodology as before, with two differences. First, we used ten fold cross validation in order to be able to perform significance tests. Second, we pooled the results across all ten folds to generate the precision recall curves. Due to space constraints, we only present results for 3 datasets: Datasets I, II, and III. The Precision/Recall curves are shown in Figures 5, 6, and 7.

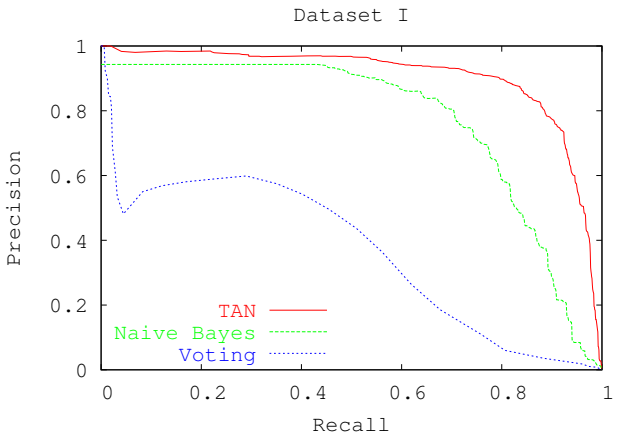


Figure 5: Precision Recall for Dataset I

Our results show much better performance for Datasets I and II. This is due to better rule quality. In this case we were able to find rules which have excellent recall, and the Bayes nets perform quite well at also achieving good precision. The results are particularly satisfactory using TAN on dataset I, as shown in Figure 5, where we can achieve precision over 60% for very high level recall. Dataset III was the hardest of all datasets for us. It shows a case where it is difficult to achieve both high precision and recall. This is because there is little information on individuals. In this case, improving recall requires trusting in only one or two rules, resulting in low precision.

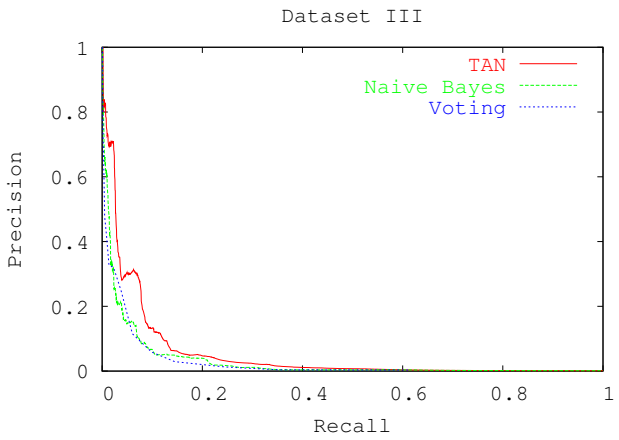


Figure 7: Precision Recall for Dataset III

The precision recall curve for the TAN algorithm dominates the curves for Naïve Bayes and ensemble voting on all six datasets. We have calculated the areas under the Precision/Recall curve for each fold in datasets I, II, III and the differences are statistically significant with 99% confidence. For each dataset, there are several places where TAN yields at least a 20 percentage point increase in precision, for the same level of recall over both Naïve Bayes and voting. On two of the six datasets, Naïve Bayes beats voting, while on the remaining four they have comparable performance. One reason for TAN's dominance compared to Naïve Bayes is the

presence of rules which are simply refinements of other rules. The TAN model is able to capture some of these interdependencies, whereas Naïve Bayes explicitly assumes that these dependencies do not exist. The Naïve Bayes independence assumption accounts for the similar performance compared to voting on several of the datasets.

In situations with imprecise rules and a preponderance of negative examples, such as link discovery domains, Bayesian models and especially TAN provide an advantage. One area where both TAN and Naïve Bayes excel is in handling imprecise rules. The Bayes nets effectively weight the precision of each rule either individually or based on the outcome of another rule in the case of TAN. The Bayesian nets further combine these probabilities to make a prediction of the final classification allowing them to discount the influence of spurious rules in the classification process. Ensemble voting does not have this flexibility and consequently lacks robustness to imprecise rules. Another area where TAN provides an advantage is when multiple imprecise rules provide significant overlapping coverage on positive examples and a low level of overlapping coverage on negative examples. The TAN network can model this scenario and weed out the false positives. One potential disadvantage to the Bayesian approach is that it could be overly cautious about classifying something as a positive. The high number of negative examples relative to the number of positive examples, and the corresponding concern of a high false positive rate, helps mitigate this potential problem. In fact, at similar levels of recall, TAN has a lower false positive rate than voting.

5 Related Work

Identity Uncertainty is a difficult problem that arises in areas such as Citation Matching, Record Linkage and De-Duplication in Databases, Natural Language Processing, in addition to Intelligence Analysis. A seminal work in this area is the theory of Record Linkage (Fellegi and Sunter 1969), based on scoring the distances between two feature vectors (using Naïve Bayes in the original work) and merging records below some threshold. Systems such as Citeseer (Lawrence et al. 1999) apply similar ideas by using text similarity. The field of record matching has received significant contributions (Monge and Elkan 1996; Cohen and Richman 2002; Buechi et al. 2003; Bilenko and Mooney 2003; Zelenko et al. 2003; Hsiung et al. 2004). On the other hand, it has been observed that interactions between identifiers can be crucial in identifying them (Morton 2000). Pasula et al. (2002) use relational probabilistic models to establish a probabilistic network of individuals, and then use Markov Chain Monte Carlo to do inference on the citation domain. McCallum and Wellner (2003) use discriminative models, Conditional Random Fields, for the same task. These approaches rely on prior understanding of the features of interest, usually text based. Such knowledge may not be available for Intelligence Analysis tasks.

Detecting features of interest was therefore our first step, and the present work fits into the popular category of using ILP for feature construction. Such work treats ILP-constructed rules as Boolean features, re-represents each example as a feature vector, and then uses a feature-vector learner to produce a final classifier. To our knowledge, the work closest to ours is the one by Pompe and Kononenko (1995), who were the first to apply Naïve Bayes to combine clauses. Other work in this category was by Srinivasan and King (1997), for the task of predicting biological activities of molecules from their atom-and-bond structures. Some other work, especially on propositionalization of First Order Logic (FOL) (Alphonse and Rouveirol 2000), has been developed that converts the training sets to propositions and then applies feature vector techniques to the converted data. This is similar to what we do, however we first learn from FOL and, then learn the network structure and parameters using the feature vectors obtained with the FOL training, resulting in much smaller feature vectors than in other work.

Our paper contributes two novel points to this category of work. First, it highlights the relationship between this category of work and ensembles in ILP, because when the feature-vector learner is Naïve Bayes the learned model can be considered a weighted vote of the rules. Second, it shows that when the features are ILP-learned rules, the independence assumption in Naïve Bayes may be violated badly enough to yield a high false positive rate. This false positive rate can be reduced by permitting strong dependencies to be explicitly noted, through learning a Tree Augmented Naïve Bayes network (TAN).

6 Conclusions and Future Work

Identity Equivalence is an important problem in Intelligence Analysis. Quite often, individuals want to hide their identities, and therefore we cannot rely on textual information. Instead, we need to use attributes and contextual information. We show that good results can be achieved by using multi-relational learning to learn rules, whose output is then combined to lower the false positive rate. We were particularly interested in Bayesian methods for the latter because they associate a probability with each prediction, which can be thought of as the classifier's confidence in the final classification. We compare how three different approaches for combining rules learned by an ILP system perform on an application where data is subject to corruption and unobservability. We demonstrate experimentally that we can significantly lower the false positive rate through rule combination schemes.

We obtained the best precision recall results in our application using a TAN network to combine rules. Precision was a major concern to us due to the high ratio of negative examples to positive examples. TAN had better precision than Naïve Bayes or unweighted voting, because it is more robust at handling redundancy between rules.

In future work we plan to experiment with different applications and Bayesian network structures. We are interested in learning rules with aggregates. We plan to further continue work based on the observation that we learn a single CLP($\mathcal{BN}$) network (Santos Costa et al. 2003). This observation suggests that a stronger coupling between the learning phases could be useful.

7 Acknowledgments

Support for this research was partially provided by U.S. Air Force grant F30602-01-2-0571. We would also like to thank Irene Ong, Bob Schrag, and Jude Shavlik for all their help. Inês Dutra and Vítor Santos Costa are on leave from Federal University of Rio de Janeiro, Brazil.

References

J. Adibi, H. Chalupsky, E. Melz, and A. Valente. The KOJAK Group Finder: Connecting the Dots via Integrated Knowledge-Based and Statistical Reasoning. In *Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-04)*, page To Appear, 2004.

E. Alphonse and C. Rouveirol. Lazy propositionalisation for relational learning. In Horn W., editor, *14th European Conference on Artificial Intelligence, (ECAI'00) Berlin, Allemagne*, pages 256–260. IOS Press, 2000.

Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

Martin Buechi, Andrew Borthwick, Adam Winkel, and Arthur Goldberg. Cluemaker: A language for approximate record matching. In *IQ*, pages 207–223, 2003.

William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD*, pages 475–480, 2002.

Jesse Davis, Vítor Santos Costa, Irene M. Ong, David Page, and Ins C. Dutra. Using Bayesian Classifiers to Combine Rules. In *3rd Workshop on Multi-Relational Data Mining*, Seattle, USA, August 2004.

I. Fellegi and A. Sunter. Theory of record linkage. *Journal of the American Stastistical Association*, 64:1183–1210, 1969.

N. Friedman, I. Nachman, and D. Pe'er. Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 206–215, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

Nir Friedman, David Geiger, and Moises Goldszmidt. Bayesian networks classifiers. *Machine Learning*, 29:131–163, 1997.

Paul Hsiung, Andrew Moore, Daniel Neill, and Jeff Schneider. Alias detection in link data sets. Master's thesis, Carnegie Mellon University, March 2004. Carnegie Mellon University.

Arno J. Knobbe, Marc de Haas, and Arno Siebes. Propositionalisation and aggregates. In *PKDD01*, pages 277–288, 2001.

N. Lavrac and S. Dzeroski, editors. *Relational Data Mining*. Springer-Verlag, Berlin, September 2001. ISBN 3-540-42289-7.

Steve Lawrence, C. Lee Giles, and Kurt D. Bollacker. Autonomous citation matching. In *Agents*, pages 392–393, 1999.

Andrew McCallum and Ben Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *IIWeb*, pages 79–84, 2003.

Alvaro E. Monge and Charles Elkan. The field matching problem: Algorithms and applications. In *KDD*, pages 267–270, 1996.

Thomas S. Morton. Coreference for NLP Applications. In *ACL*, 2000.

S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay. Learning relational probability trees. In *KDD '03*, pages 625–630. ACM Press, 2003. ISBN 1-58113-737-0.

Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart J. Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In *NIPS*, pages 1401–1408, 2002.

Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *KDD '03*, pages 167–176, 2003. ISBN 1-58113-737-0.

U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 417–436. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

Vítor Santos Costa, David Page, Maleeha Qazi, and James Cussens. CLP($\mathcal{BN}$): Constraint Logic Programming for Probabilistic Knowledge. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI03)*, pages 517–524, Acapulco, Mexico, August 2003.

Robert C. Schrag. EAGLE Y2.5 Performance Evaluation Laboratory (PE Lab) Documentation Version 1.5. Internal report, Information Extraction & Transport Inc., April 2004.

A. Srinivasan. *The Aleph Manual*, 2001.

A. Srinivasan and R. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton, editor, *Proceedings of the Sixth Inductive Logic Programming Workshop*, LNAI 1314, pages 89–104, Berlin, 1997. Springer-Verlag.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106, 2003.

View Learning for Statistical Relational Learning: With an Application to Mammography

Jesse Davis, Elizabeth Burnside, Inês Dutra, David Page,
Raghu Ramakrishnan, Vítor Santos Costa, Jude Shavlik
University of Wisconsin - Madison
1210 West Dayton
Madison, WI 53706, USA
email: jdavis@cs.wisc.edu

Abstract

Statistical relational learning (SRL) constructs probabilistic models from relational databases. A key capability of SRL is the learning of arcs (in the Bayes net sense) connecting entries in different rows of a relational table, or in different tables. Nevertheless, SRL approaches currently are constrained to use the existing database schema. For many database applications, users find it profitable to define alternative "views" of the database, in effect defining new fields or tables. Such new fields or tables can also be highly useful in learning. We provide SRL with the capability of learning new views.

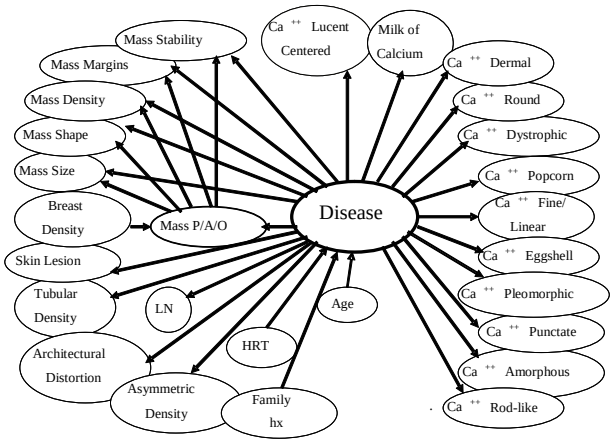


Figure 1: Expert Bayes Net

1 Introduction

Statistical Relational Learning (SRL) focuses on algorithms for learning statistical models from relational databases. SRL advances beyond Bayesian network learning and related techniques by handling domains with multiple tables, representing relationships between different rows of the same table, and integrating data from several distinct databases. SRL techniques currently can learn joint probability distributions over the fields of a relational database with multiple tables. Nevertheless, they are constrained to use only the tables and fields already in the database, without modification. Many human users of relational databases find it beneficial to define alternative *views* of a database—further fields or tables that can be computed from existing ones. This paper shows that SRL algorithms also can benefit from the ability to define new views, which can be used for more accurate prediction of important fields in the original database.

We will augment SRL algorithms by adding the ability to learn new fields, intensionally defined in terms of existing fields and intensional background knowledge. In database terminology, these new fields constitute a learned *view* of the database. We use Inductive Logic Programming (ILP) to learn rules which intensionally define the new fields.

We test the approach in the specific application of creating an expert system in mammography. We chose this application for a number of reasons. First, it is an important practical application with sizable data. Second, we have access to an expert developed system. This provides a base reference against which we can evaluate our work. Third, a large proportion of examples are negative. This distribution skew is often found in multi-relational applications. Last, our data consists of a single table. This allows us to compare our techniques against standard propositional learning. In this case, it is sufficient for view learning to extend an existing table with new fields. It should be clear that for other applications the approach can yield additional tables.

2 View Learning for Mammography

Offering breast cancer screening to the ever-increasing number of women over age 40 represents a great challenge. Cost-effective delivery of mammography screening depends on a consistent balance of high sensitivity and high specificity. Recent articles demonstrate that subspecialist, expert mammographers achieve this balance and perform significantly better than general radiologists [24; 1]. General radiologists have higher false positive rates and hence biopsy rates, diminishing the positive predictive value for mammography [24; 1]. Despite the fact that specially trained mammographers detect breast cancer more accurately, there is a longstanding shortage of these individuals [6].

An expert system in mammography has the potential to help the general radiologist approach the effectiveness of a

subspecialty expert, thereby minimizing both false negative and false positive results.

Bayesian networks (BNs) are probabilistic graphical models that have been applied to the task of breast cancer diagnosis from mammography data [12; 2; 3]. BNs produce diagnoses with probabilities attached. Because of their graphical nature, they are comprehensible to humans and useful for training. As an example, Figure 1 shows the structure of a Bayesian network developed by a subspecialist, expert mammographer. For each variable (node) in the graph, the Bayes net has a conditional probability table giving the probability distribution over the values that variable can take for each possible setting of its parents. The Bayesian network in Figure 1 achieves accuracies higher than that of other systems and of general radiologists who perform mammograms, and commensurate with the performance of radiologists who specialize in mammography [2].

Figure 2 shows the main table (with some fields omitted for brevity) in a large relational database of mammography abnormalities. Data was collected using the National Mammography Database (NMD) standard established by the American College of Radiology. The NMD was designed to standardize data collection for mammography practices in the US and is widely used for quality assurance. Figure 2 also presents a hierarchy of the types of learning that might be used for this task. Level 1 and Level 2 are standard types of Bayesian network learning. Level 1 is simply learning the parameters for an expert-supplied network structure. Level 2 involves learning the actual structure of the network in addition to its parameters.

Notice that to predict the probability of malignancy of an abnormality, the Bayes net uses only the record for that abnormality. Nevertheless, data in other rows of the table may also be relevant: radiologists may also consider other abnormalities on the same mammogram or previous mammograms. For example, it may be useful to know that the same mammogram also contains another abnormality, with a particular size and shape; or that the same person had a previous mammogram with certain characteristics. Incorporating data from other rows in the table is not possible with existing Bayesian network learning algorithms and requires statistical relational learning (SRL) techniques, such as probabilistic relational models [8]. Level 3 in Figure 2 shows the state-of-the-art in SRL techniques, illustrating how relevant fields from other rows (or other tables) can be incorporated in the network, using aggregation if necessary. Rather than using only the size of the abnormality under consideration, the new aggregate field allows the Bayes net to also consider the average size of all abnormalities found in the mammogram.

Presently, SRL is limited to using the original view of the database, that is, the original tables and fields, possibly with aggregation. Despite the utility of aggregation, simply considering only the existing fields may be insufficient for accurate prediction of malignancies. Level 4 in Figure 2 shows the key capability that will be introduced and evaluated in this paper: Using techniques from rule learning to learn a new *view*. The new view includes two new features utilized by the Bayes net that cannot be defined simply by aggregation of existing features. The new features are defined by two learned rules that capture "hidden" concepts central to accurately predicting malignancy, but that are not explicit in the given database tables. One learned rule states that a change in the shape of the abnormality at a location since an earlier mammogram may be indicative of a malignancy. The other says that an *increase* in the average of the sizes of the abnormalities may be indicative of malignancy. Note that both rules require reference to other rows in the table for the given patient, as well as intensional background knowledge to define concepts such as "increases over time." Neither rule can be captured by standard aggregation of existing fields.

3 View Learning Framework

One can imagine a variety of approaches to perform view learning. Our closing section discusses a number of alternatives, including performing view learning and structure learning at the same time, in the same search. For the present work, we apply existing technology in a new fashion to obtain a view learning capability.

Any relational database can be naturally and simply represented in a subset of first-order logic [21]. Inductive logic programming (ILP) provides algorithms to learn rules, also expressed in logic, from such relational data [15], possibly together with background knowledge expressed as a logic program. ILP systems operate by searching a space of possible logical rules, looking for rules that score well according to some measure of fit to the data. We use ILP to learn rules to predict whether an abnormality is malignant. We treat each rule as an additional binary feature; true if the body, or condition, of the rule is satisfied, and otherwise false. We then run the Bayesian network structure learning algorithm, allowing it to use these new features in addition to the original features. Below is a simple rule, covering 48 positive examples and 123 negative examples:

```
Abnormality A in mammogram M
        may be malignant if:
   A's tissue is not asymmetric,
   M contains another abnormality A2,
   A2's margins are spiculated, and
   A2 has no architectural distortion.
```

This rule can now be used as a field in a new view of the database, and consequently as a new feature in the Bayesian network. The last two lines of the rule refer to other rows of the relational table for abnormalities in the database. Hence this rule encodes information not available to the current version of the Bayesian network.

4 Experiments

The purposes of the experiments we conducted are two-fold. First, we want to determine if using SRL yields an improvement compared to propositional learning. Secondly, we want to evaluate whether using Inductive Logic Programming (ILP) to create features, which embody a new "view" of the database, adds a benefit over current SRL algorithms. We looked at adding two types of relational attributes, aggregate features and horn-clause (ILP) rules. Aggregate features represent summaries of abnormalities found either in a particular mammogram or for a particular patient. We performed

Patient	Abnormality	Date	Mass Shape	…	Mass Size	Location	Benign/Malign
P1	1	5/02	Spic		0.03	RU4	B
P1	2	5/04	Var		0.04	RU4	M
P1	3	5/04	Spic		0.04	LL3	B
…	…	…	…		…	…	…

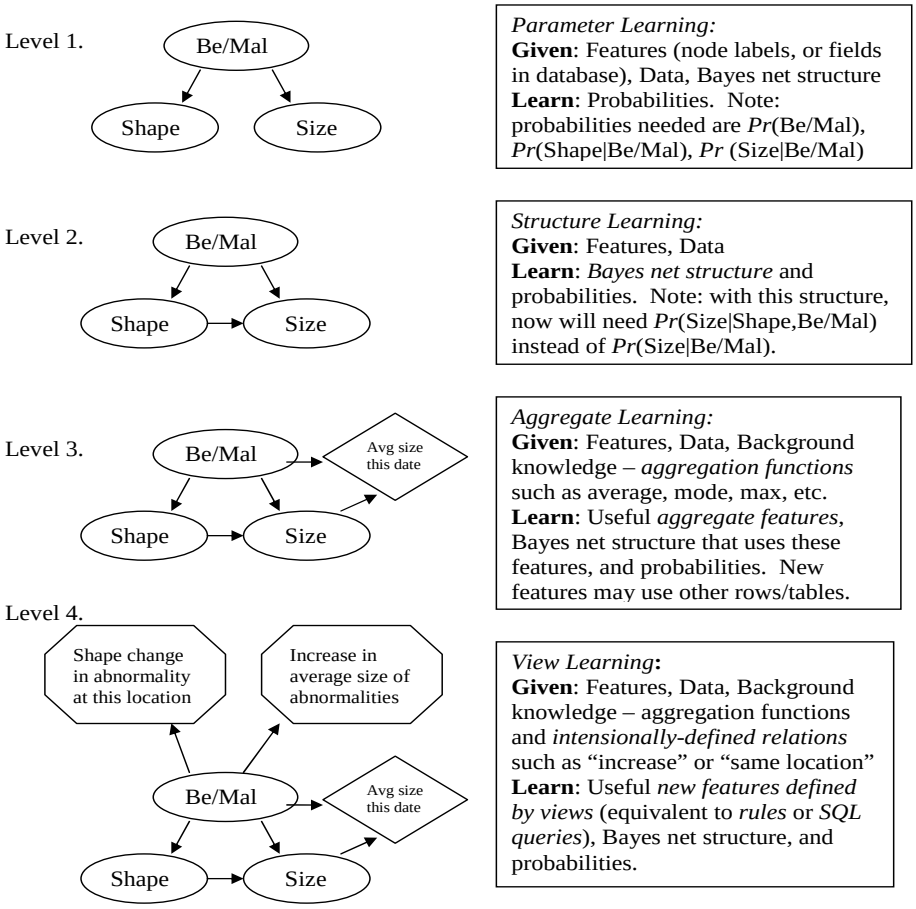


Figure 2: Hierarchy of learning types. Levels 1 and 2 are available through ordinary Bayesian network learning algorithms, Level 3 is available only through state-of-the-art SRL techniques, and Level 4 is described in this paper.

a series of experiments, aimed at discovering if moving up in the hierarchy outlined in Figure 2 would improve performance. First, we tried to learn a structure with just the original attributes which performed better than the expert structure. This corresponds to Level 2 learning. Next, we added aggregate features to our network. Finally, we created new features using ILP. We investigated adding the features proposed by the ILP system as well as the aggregate to the network.

We experimented with a number of structure learning algorithms for the Bayesian Networks, including Naïve Bayes, Tree Augmented Naïve Bayes [7], and the sparse candidate algorithm [9]. However, we obtained best results with the TAN algorithm in all experiments, so we will focus our discussion on TAN. In a TAN network, each attribute can have at most one other parent in addition to the class variable. The TAN model can be constructed in polynomial time with a guarantee that the model maximizes the Log Likelihood of the network structure given the dataset [10; 7].

4.1 Methodology

The dataset contains 435 malignant abnormalities and 65365 benign abnormalities. To evaluate and compare these approaches, we used stratified 10-fold cross-validation. We randomly divided the abnormalities into 10 roughly equal-sized sets, each with approximately one-tenth of the malignant abnormalities and one-tenth of the benign abnormalities. When evaluating just the structure learning and aggregation, 9 folds were used for the training set. When performing aggregation, we used binning to discretize the created features. We took care to only use the examples in the train set to determine the cut bin widths. When performing "view learning", we had

two steps in the learning process. In the first part, 4 folds of data were used to learn the ILP rules. Afterwards,the remaining 5 folds were used to learn the Bayes net structure and parameters.

When using cross-validation on a relational database, there exists one major methodological pitfall. Some of the cases may be related. For example, we may have multiple abnormalities for a single patient. Because these abnormalities are related (same patient), having some of these in the training set and others in the test set may cause us to perform better on those test cases than we would expect to perform on cases for other patients. To avoid such "leakage" of information into a training set, we ensured that all abnormalities associated with a particular patient are placed into the same fold for cross-validation. Another pitfall is that we may learn a rule that predicts an abnormality to be malignant based on properties of abnormalities in *later* mammograms. We never predict the status of an abnormality at a given date based on findings recorded with later dates.

We present the results of our first experiment using both ROC and precision recall curves. Because of our skewed class distribution, or large number of benign cases, we prefer precision-recall curves over ROC curves because they better show the number of "false alarms," or unnecessary biopsies. Therefore, we use precision-recall curves for the remainder of results. Here precision is the percentage of abnormalities that we classified as malignant that are truly cancerous. Recall is the percentage of malignant abnormalities that were correctly classified. To generate the curves, we pooled the results over all ten folds in the following manner. We treat each prediction as if it had been generated from the same model. We sorted the estimates and used all possible split points to create the graphs.

4.2 Results

The relational database containing the mammography data contains one row for each abnormality in a mammogram. Fields in this relational table include all those shown in the Bayesian network of Figure 1. Therefore it is straightforward to use existing Bayesian network structure learning algorithms to learn a possibly improved structure for the Bayesian network. We compared the performance of the best learned networks against the expert defined structure shown in Figure 1. We estimated the parameters of the expert structure from the dataset using maximum likelihood estimates with Laplace correction. Figure 3 shows the ROC curve for these experiments, and Figure 4 shows the Precision-Recall curves. Figure 7 shows the area under the precision-recall curve for the expert network (L1) and with learned structure (L2). We only consider recalls above 50%, as for this application radiologists would be required to perform at least at this level. We further use the paired t-test to compare the areas under the curve for every fold. We found the difference to be statistically significant with a 99% level of confidence.

With the help of a radiologist, we selected the numeric and ordered features in the database and computed aggregates for each of these features. We determined that 27 of the 36 attributes were suitable for aggregation. We computed aggregates on both the patient and the mammogram level. On the

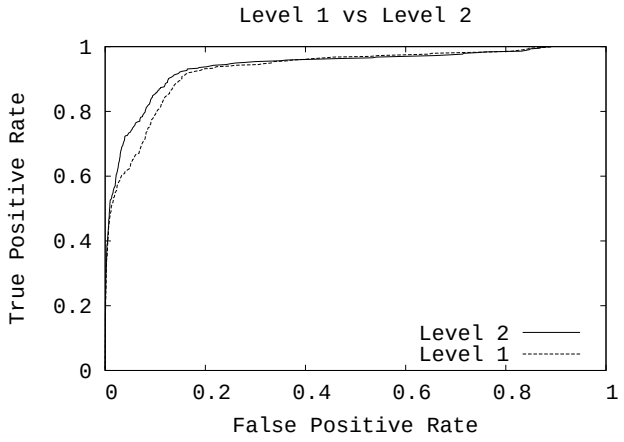


Figure 3: ROC Curves for Structure Learning. (Level 2)

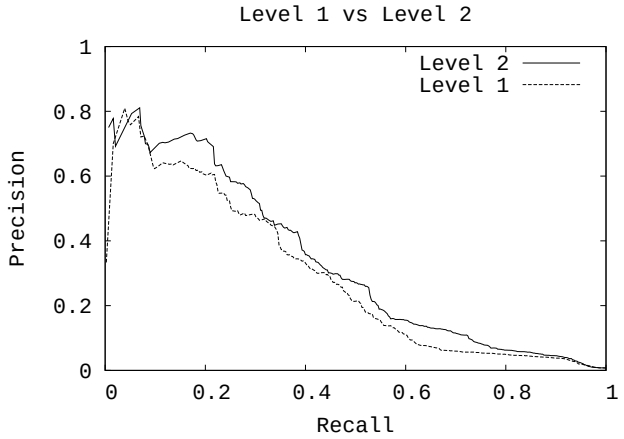


Figure 4: Precision Recall Curves for Structure Learning. (Level 2)

patient level, we looked at all of the abnormalities for a specific patient. On the mammogram level, we only considered the abnormalities present on that specific mammogram. To discretize the averages, we divided each range into three bins. For binary features we had predefined bin sizes, while for the other features we attempted to get equal numbers of abnormalities in each bin. For aggregations functions we used maximum and average. The aggregation introduced 108 new features. For the interested reader, the following paragraph presents further details of our aggregation process.

We used a three step process to construct aggregate features. First, we chose a field to aggregate. Second, we selected an aggregation function. Third, we needed to decide over which rows to aggregate the feature, that is, which keys or links to follow. This is known as a slot chain in PRM terminology. In our database, two such links exist. The patient id field allows access to all the abnormalities for a given patient, providing aggregation on the patient level. The second key is the combination of patient id and mammogram date, which returns all abnormalities for a patient on a specific mammogram, providing aggregation on the mammogram level. To

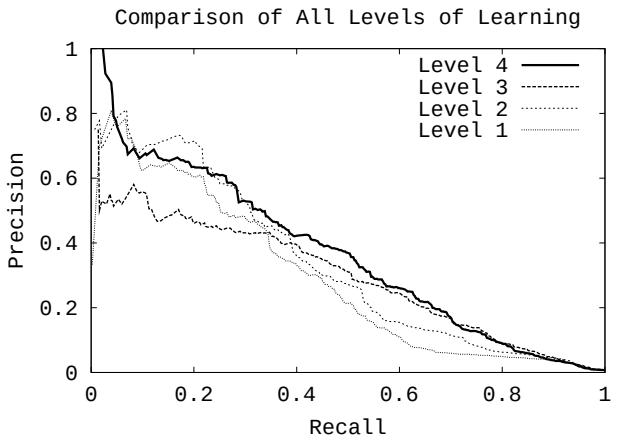


Figure 6: Precision Recall Curves for Each Level of Learning

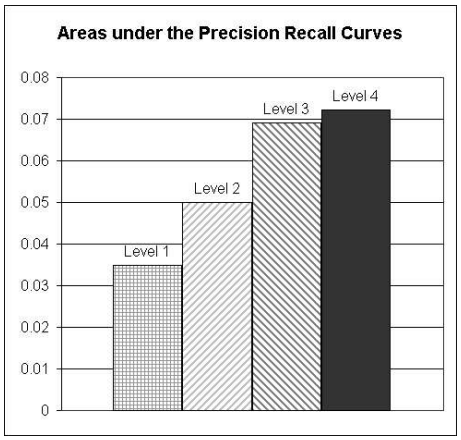


Figure 7: Area Under the Curve For Recalls Above 50%

demonstrate this process we will work though an example of computing an aggregate feature for patient 1 in the database given in Figure 2. We will aggregate on the Mass Size field and use average as the aggregation function. Patient 1 has three abnormalities, one from a mammogram in May 2002 and two on a mammogram from May 2004. To calculate the aggregate on the patient level, we would average the size for all three abnormalities, which is .0367. To find the aggregate on the mammogram level for patient 1, he have to perform two separate computations. First, we follow the link P1 and 5/02, which yields abnormality 1. The average for this key mammogram is simply .03. Second, we follow the link P1 and 5/04, which yields abnormalities 2 and 3. The average for these abnormalities is .04. Figure 5 shows the database following .

Next we tested whether useful new fields could be computed by rule learning. Specifically, we used the ILP system Aleph [26] to learn rules predictive of malignancy. Several thousand distinct rules were learned for each fold, with each rule covering many more malignant cases than (incorrectly covering) benign cases. In order to obtain a varied set of rules, we ran Aleph using every positive example in each fold served as a seed for the search. We avoid the rule overfitting found by other authors [18] by doing breadth-first search for rules and by having a minimal limit on coverage. Each seed generated anywhere from zero to tens of thousands of rules. We post processed the rules using a greedy algorithm, where we select the best scoring rule that covers new examples first. For each fold, the 50 best clauses were selected based on 3 criteria: (1) they needed to be multi-relational; (2) they needed to be distinct; (3) they needed to cover a significant number of malignant cases. The resulting views were added as new features to the database. Figure 6 includes a comparison of all levels of learning.

We can observe very significant improvements when adding multi-relational features. Both rules and aggregates achieved better performance. Aggregates do better for higher recalls, while rules do better for medium recalls. We believe this is because ILP rules are more accurate than the other features, but have limited coverage.

Level 4 performs as well as aggregates for high recalls, and close to ILP for medium recalls. According to the paired t-test the improvement of Level 4 over Level 2 is significant, using the area under the curve metric, at the 99% level. Meanwhile, Level 3 presents an improvement over Level 2, using the area under the curve metric, at the 97% confidence level.

Levels 1 and 2 correspond to standard propositional learning whereas levels 3 and 4 incorporate relational information. In this task, considering relational information is crucial for improving preformance. Furthermore, the process of generating the views in Level 4 has been useful to the radiologist as it has potentially identified novel correlations between attributes.

5 Related Work

Research in SRL has advanced along two main lines, methods that allow graphical models to represent relations and frameworks that extend logic to handle probabilities. Along the first line, probabilistic relational models, or PRMs, introduced by Friedman, Getoor, Koller and Pfeffer, represent one of the first attempts to learn the structure of graphical models while incorporating relational information[8]. Recently Heckerman, Meek and Koller have discussed extensions to PRMs and compared them to to other graphical models[11]. A statistical learning algorithm for probabilitistic logic representations was first given by Sato [23] and later, Cussens [5] proposed a more general algorithm to handle log linear models. Additionally, Muggleton [16] has provided learning algorithms for stochastic logic programs. The structure of the logic program is learned using ILP techniques, while the parameters are learned using an algorithm scaled up from that used for stochastic context-free grammars.

Newer representations garnering arguably the most attention are Bayesian logic programs [13] (BLPs), constraint logic programming with Bayes net constraints, or CLP($\mathcal{BN}$) [4], and Markov Logic Networks (MLNs) [22]. Markov Logic Networks are most similar to our approach. Nodes of MLNs are the ground instances of the literals in the rule, and the arcs correspond to the rules. One major difference is that in our approach nodes are the rules themselves. Although we

Patient	Abnormality	Date	Mass Shape	...	Mass Size	Location	Average Patient Mass Size	Average Mammogram Mass Size	Be/Mal
P1	1	5/02	Spic	...	0.03	RU4	0.0367	0.03	B
P1	2	5/04	Var	...	0.04	RU4	0.0367	0.04	M
P1	3	5/04	Spic	...	0.04	LL4	0.0367	0.04	B
...	...	...	...	...	...	...	...	...	...

Figure 5: Database after Aggregation on Mass Size Field

cannot work at the same level of detail, our approach makes it straightforward to combine logical rules with other features, and we now can take full advantage of propositional learning algorithms.

The present work builds upon previous work on using ILP for feature construction. Such work treats ILP-constructed rules as Boolean features, re-represents each example as a feature vector, and then uses a feature-vector learner to produce a final classifier. To our knowledge, Pompe and Kononenko [19] were the first to apply Naïve Bayes to combine clauses. Other work in this category was by Srinivasan and King [25], who use rules as extra features for the task of predicting biological activities of molecules from their atom-and-bond structures. Popescul et.al. [20] use $k - means$ to derive cluster relations, which are then combined with the original features through structural regression. In a different vein, Relational Decision Trees [17] use aggregation to provide extra features on a multi-relational setting, and are close to our Level 3 setting. Knobbe et al. [14] proposed numeric aggregates in combination with logic-based feature construction for single attributes. Perlich and Provost discuss several approaches for attribute construction using aggregates over multi-relational features [18]. The authors also propose a hierarchy of levels of learning: feature vectors, independent attributes on a table, multidimensional aggregation on a table, and aggregation across tables. Some of these techniques in their hierarchy could be applied to perform view learning in SRL.

6 Conclusions and Future Work

We presented a method for statistical relational learning which integrates learning from attributes, aggregates, and rules. Our example application shows benefits from the several levels of learning we proposed. Level 2, structure learning, clearly outperforms the expert structure. We further show that multi-relational techniques can achieve very significant improvements, even on a single table domain, and that the most consistent improvement is obtained by using Level 4, both aggregates and new views.

We believe that further improvements are possible. It makes sense to include aggregates in the background knowledge for rule generation. Alternatively, one can extend rules with aggregation operators, as proposed in recent work by Vens et al. [27]. We have found the rule selection problem to be non-trivial. Our greedy algorithm often generates too similar rules, and is not guaranteed to maximize coverage. We would like to approach this problem as an optimization problem weighing coverage, diversity, and accuracy.

Our approach of using ILP to learn new features for an existing table merely scratches the surface of the potential for view learning. A more ambitious approach would be to more closely integrate structure learning and view learning. A search could be performed in which each "move" in the search space is either to modify the probabilistic model or to refine the intensional definition of some field in the new view. Going further still, one might learn an intensional definition for an entirely new table. As a concrete example, for mammography one could learn rules defining a binary predicate that identifies "similar" abnormalities. Because such a predicate would represent a many-to-many relationship among abnormalities, a new table would be required.

7 Acknowledgments

Support for this research was partially provided by U.S. Air Force grant F30602-01-2-0571. Elizabeth Burnside is supported by a General Electric Research in Radiology Academic Fellowship. Inês Dutra is on leave from Federal University of Rio de Janeiro, Brazil. Vítor Santos Costa is on leave from the University of Porto, Portugal and the Federal University of Rio de Janeiro, Brazil. We would like to thank Rich Maclin for reading over several drafts of this paper. We would also like to thank the referees for their insightful comments.

References

[1] M.L. Brown, F. Houn, E.A. Sickles, and L.G. Kessler. Screening mammography in community practice: positive predictive value of abnormal findings and yield of

follow-up diagnostic procedures. *AJR Am J Roentgenol*, 165:1373–1377, 1995.

[2] E.S. Burnside, D.L. Rubin, and R.D. Shachter. A Bayesian network for screening mammography. In *AMIA*, pages 106–110, 2000.

[3] E.S. Burnside, D.L. Rubin, and R.D. Shachter. Using a Bayesian network to predict the probability and type of breast cancer represented by microcalcifications on mammography. *Medinfo*, 2004:13–17, 2004.

[4] V. Santos Costa, D. Page, M. Qazi, and J. Cussens. CLP($\mathcal{BN}$): Constraint logic programming for probabilistic knowledge. In *UAI-03*, pages 517–524, Acapulco, 2003.

[5] J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.

[6] G. Ecklund. Shortage of qualified breast imagers could lead to crisis. *Diagn Imaging*, 22:31–33, 2000.

[7] Nir Friedman, David Geiger, and Moises Goldszmidt. Bayesian networks classifiers. *Machine Learning*, 29:131–163, 1997.

[8] Nir Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Stockholm, Sweden, 1999.

[9] Nir Friedman, I. Nachman, and D. Pe'er. Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. In *UAI-99*, pages 206–215, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[10] Dan Geiger. An entropy-based learning algorithm of Bayesian conditional trees. In *UAI-92*, pages 92–97, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[11] D Heckerman, C Meek, and D Koller. Probabilistic Entity-Relationship Models, PRMs, and Plate Models, Technical Report MSR-TR-2004-30, Microsoft Research. Technical report, Microsoft Research, 2004.

[12] C.E. Kahn Jr, L.M. Roberts, K.A. Shaffer, and P. Haddawy. Construction of a Bayesian network for mammographic diagnosis of breast cancer. *Comput Biol Med.*, 27:19–29, 1997.

[13] K. Kersting and L. De Raedt. Basic principles of learning Bayesian logic programs, 2002.

[14] Arno J. Knobbe, Marc de Haas, and Arno Siebes. Propositionalisation and aggregates. In *PKDD01*, pages 277–288, 2001.

[15] S.H. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8:295–318, 1991.

[16] S.H. Muggleton. Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence*, 4(041), 2000.

[17] Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay. Learning relational probability trees. In *KDD '03*, pages 625–630. ACM Press, 2003.

[18] Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *KDD '03*, pages 167–176, 2003.

[19] U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *ILP95*, pages 417–436, 1995.

[20] Alexandrin Popescul, Lyle H. Ungar, Steve Lawrence, and David M. Pennock. Statistical relational learning for document mining. In *ICDM03*, pages 275–282, 2003.

[21] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw Hill, 2000.

[22] Matt Richardson and Pedro Domingos. Markov logic networks. http://www.cs.washington.edu/homes/pedrod/kbmn.pdf, 2004.

[23] T. Sato. A statistical learning method for logic programs with distributional semantics. In L. Sterling, editor, *Proceedings of the Twelth International conference on logic programming*, pages 715–729, Cambridge, Massachusetts, 1995. MIT Press.

[24] E.A. Sickles, D.E. Wolverton, and K.E. Dee. Performance parameters for screening and diagnostic mammography: specialist and general radiologists. *Radiology*, 224:861–869, 2002.

[25] A. Srinivasan and R. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In *ILP97*, pages 89–104, 1997.

[26] Ashwin Srinivasan. *The Aleph Manual*, 2001.

[27] Celine Vens, Anneleen Van Assche, Hendrik Blockeel, and Sašo Džeroski. First order random forests with complex aggregates. In *ILP*, pages 323–340, 2004.

Learning to Extract Genic Interactions Using Gleaner

Mark Goadrich RICHM@CS.WISC.EDU
Louis Oliphant OLIPHANT@CS.WISC.EDU
Jude Shavlik SHAVLIK@CS.WISC.EDU
Department of Biostatistics and Medical Informatics and Department of Computer Sciences, University of Wisconsin-Madison, 1300 University Avenue, Madison, WI, 53706 USA

Abstract

We explore here the application of Gleaner, an Inductive Logic Programming approach to learning in highly-skewed domains, to the Learning Language in Logic 2005 biomedical information-extraction challenge task. We create and describe a large number of background knowledge predicates suited for this task. We find that Gleaner outperforms standard Aleph theories with respect to recall and that additional linguistic background knowledge improves recall.

1. Introduction

Information Extraction (IE) is the process of scanning unstructured text for objects of interest and facts about these objects. Recently, biomedical journal articles have been a major source of interest in the IE community for a number of reasons: the amount of data available is enormous; the objects, proteins and genes, do not have standard naming conventions; and there is interest from biomedical practitioners to quickly find relevant information (Blaschke et al., 2002, Shatkay & Feldman, 2003, Eliassi-Rad & Shavlik, 2001, Ray & Craven, 2001, Bunescu et al., 2004).

IE can be framed as a machine learning task: given information in unstructured text documents, extract the relevant objects and relationships between them. We believe that Inductive Logic Programming (ILP) is well-suited for IE in biomedical domains. ILP offers the advantages of (1) a straight-forward way to incorporate domain knowledge and expert advice and (2) produces logical clauses suitable for analysis and revision by humans to improve performance.

Appearing in *Proceedings of the 4th Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

In this article, we report both the data-preparation techniques and the results of applying Gleaner (Goadrich et al., 2004) to the Learning Language in Logic 2005 biomedical information extraction task of learning genic interactions. Gleaner is a two-stage ILP algorithm that (1) learns a broad spectrum of clauses and (2) then combines them into a thresholded disjunctive clause aimed at maximizing precision for a particular choice of recall. We compare our results to standard Aleph (Srinivasan, 2003) using recall and precision, and discuss areas open to future research.

2. Data Preparation

Our dataset for this article is the Learning Language in Logic challenge task[1], where the goal is to learn to recognize the interaction in English sentences between protein agents and their gene targets in *Bacillus subtilis*. Sentences in the training set contained either a direct reference between an agent and a target, such as "GerE stimulates cotD transcription," or an indirect reference, such as "GerE binds to a site on one of these promoters, cotX [...]," where the relation between GerE and cotX is mediated by the phrase "these promoters." The organizers call these two subtasks *without co-reference* and *with co-reference* and we chose to learn on them separately, first learning only relationships without co-reference, and second learning only relationships with co-reference.

The training data consist of 80 sentences found in the Medline[2] database, and contain 106 relations without co-reference and 59 relations with co-reference. For each subtask, we used the other trainset as our tuneset to find an appropriate threshold for making testset predictions. While they are slightly different tasks, we found that the benefit of more examples outweighed dividing the training sets into subfolds.

[1] http://genome.jouy.inra.fr/texte/LLLchallenge/
[2] http://www.ncbi.nlm.nih.gov/pubmed

2.1. Example Filtering

Positive examples for this dataset, consisting of word/word pairings, have been labeled by the challenge-task committee, while negative examples were left up to the participants. We define negative examples on a per-sentence basis by first finding all words which participate in a positive relationship. The pairings among these words which are not labeled as positives are used as negatives for training and tuning. This produced 414 without co-reference negative examples and 261 with co-reference negative examples.

The testset was provided to us unlabeled, and contained sentences for both the task with co-reference and the task without co-reference. Unlike the training data, the testset also contained sentences which did not contain any relations. For the testset, we created examples from the pairing of all possible protein and gene names found in a provided dictionary. This produced 936 total testset examples. In subsequent experiments, we reduced this to 618 examples by removing testset examples where the agent and target of the relation were identical (since this never happened in the trainset). Ultimately there were 54 positive and 410 negative test examples for the without co-reference task and 29 positive and 384 negative test examples for the with co-reference task.

2.2. Background Knowledge

To prepare the data for learning via Inductive Logic Programming, we constructed a variety of background knowledge from sentence structure, statistical word frequencies, lexical properties, and biomedical dictionaries, examples of which can be seen in Table 1.

Our first set of relations comes from the sentence structure. We use the Brill tagger (1995) retrained on the GENIA dataset (Kim et al., 2003) to predict the part of speech for each word. Then we employ a shallow parser created by Burr Settles that uses Conditional Random Fields (Lafferty et al., 2001) trained on a standard corpus (Sang, 2001) to derive a flat parse tree, such that there are no nested phrases, for all sentences in our dataset. All phrases have the sentence as the root, and therefore all words are only members of one phrase. Figure 1 shows a sample sentence parse divided into one level of phrases.

Each word, phrase, and sentence is given a unique identifier based on its ordering within the given abstract, such as `ab11011148_sen1_ph2_w1`. This allows us to create relations between sentences, phrases and words based not on the actual text of the document but on its structure, such as `sentence_child`, `phrase_previous` and `word_next` about the tree structure and sequence of words, and predicates like `nounPhrase`, `article`, and `verb` to describe the part of speech structure. To include the actual text of the sentence in our background knowledge, the predicate `word_ID_to_string` maps these identifiers to the words. In addition, the words of the sentence are stemmed using the Porter stemmer (Porter, 1980), and currently we only use the stemmed version of words.

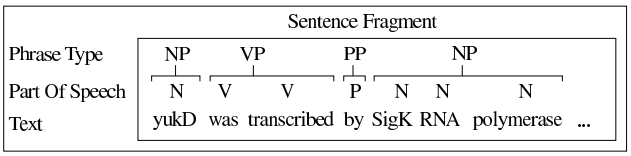


Figure 1. Sample Sentence Parse (N=noun, V=verb, P=preposition, NP=noun phrase, VP=verb phrase, PP=prepositional phrase)

General sentence-structure predicates like `word_before` and `phrase_after` are added, allowing navigation around the parse tree. Phrases are also tagged as being the first or last phrase in the sentence, likewise for words. The length of phrases is calculated and explicitly turned into a predicate, as well as the length (by words and phrases) of sentences. Also, phrases are classified as short, medium or long. An additional piece of useful information is the predicate `different_phrases`, which is true when its two arguments are distinct phrases.

Another group of background relations comes from looking at the frequency of words appearing in the target phrases in the training set. We believe these frequently occurring words could be indicators of some underlying semantic class and will be helpful for identifying correct phrases in the testset. We created Boolean predicates for several ratios - 2 times, 5 times and 10 times the general word frequency across all sentences in a given training set - using the following formula to determine which words matched which ratios:

$$\frac{P(w_i = word | w_i \in Target\ Phrase)}{P(w_i = word | w_i \notin Target\ Phrase)}$$

For example, the words "depend," "bind," and "protein" are at least 5 times more likely to appear in protein phrases than in phrases in general in the without co-reference training set. These gradations are calculated for both target arguments, protein and gene. We automatically create semantic classes consisting of these high frequency words. These semantic classes are then used to mark up all occurrences of these words in a given training and testing set.

A third source of background knowledge is de-

Table 1. Translation from Sample Sentence "ykuD was transcribed by SigK RNA polymerase from T4 of sporulation," to Prolog. This sentence is from the abstract whose PubMed ID is 11011148. Not all predicates created are listed.

Background Knowledge	Some of the Prolog Predicates Created

Sentence Structure	`sentence(ab11011148_sen4).`
	`phrase(ab11011148_sen4_ph0).`
	`phrase(ab11011148_sen4_ph1).`
	`word(ab11011148_sen4_ph0_w0).`
	`word(ab11011148_sen4_ph1_w1).`
	`word(ab11011148_sen4_ph1_w2).`
	`phrase_child(ab11011148_sen4_ph0, ab11011148_sen4_ph0_w0).`
	`word_next(ab11011148_sen4_ph0_w0, ab11011148_sen4_ph0_w1).`
	`word_ID_to_string(ab11011148_sen4_ph1_w1, 'ykuD').`
	`target_arg2_before_target_arg1(ab11011148_sen4).`

Part Of Speech	`np_segment(ab11011148_sen4_ph0).`
	`vp_segment(ab11011148_sen4_ph1).`
	`n(ab11011148_sen4_ph0_w0).`
	`v(ab11011148_sen4_ph1_w1).`
	`prep(ab11011148_sen4_ph1_w3).`

Medical Ontologies	`phrase_contains_mesh_term(ab11011148_sen4_ph3, 'RNA').`

Lexical Properties	`phrase_contains_alphanumeric_word(ab11011148_sen4_ph5).`
	`phrase_contains_specific_word(ab11011148_sen4_ph1, 'transcribed').`
	`phrase_contains_originally_leading_cap(ab11011148_sen4_ph3).`

Word Frequency	`phrase_contains_some_arg_2x_word(ab11011148_sen4_ph3).`

rived from the lexical properties of each word. `Alphanumeric` words contain both numbers and alphabetic characters, (such as "sigma 32" and "Spo0A~P") whereas `alphabetic` words have only alphabetic characters. Other lexical and morphological features include `singleChar` ("a"), `hyphenated` ("membrane-bound") and `capitalized` ("RNA"). Also, words are classified as `novelWord` ("sporulation") if they do not appear in the standard `/usr/dict/words` dictionary in UNIX. Lexical predicates are then augmented to make them more applicable to the phrase level and therefore more general. These predicates are also created for pairs and triplets of words, so we can assert that a phrase has the word "bind" tagged as a verb all in one step when we search the hypothesis space.

For our fourth source, we incorporate semantic knowledge about biology and medicine into our background relations by using the Medical Subject Headings (MeSH)[3]. As we did for the sentence structure, we have simplified this hierarchy to only be one level. Phrases are labeled with the predicate `phrase_contains_mesh_term` if any of the words in the given phrase match any words in MeSH.

[3]http://www.nlm.nih.gov/mesh/meshhome.html

Additionally, predicates are added to denote the ordering between the phrases. `Target_arg1_before_target_arg2` asserts that the protein phrase occurs before the gene phrase, similarly for `target_arg2_before_target_arg1`. Also created are `identical_target_args` (which is true when the protein and gene phrases are the same phrase, such as the phrase "sigmaB dependent katX expression") and `adjacent_target_args` (which says the adjacent phrases contains both the gene and protein), as well as the count of phrases before and after the target arguments. Overall, we defined 215 predicates for use in describing the training examples.

2.3. Enriched Data

Background knowledge was also provided by the challenge task organizers. They processed the corpus with Link Parser (Temperly et al., 1999), a tool for automatically constructing a syntactic parse tree, and refined the output to create two type of additional information. First, each word was assigned its root word, called a *lemma*. For instance, the word "are" would have the lemma "be." The second type of information was the syntactic relations between words. This included appositive, complement, modifier, negation,

Table 2. Pseudo-code for Gleaner Algorithm

Initialize Bins:
Create B recall bins, $bin_{\frac{1}{B}}, bin_{\frac{2}{B}}, ..., bin_1$,
to uniformly divide the recall range [0,1]

Populate Bins:
For $i = 1$ to K until N clauses are generated
  Pick seed example to find bottom clause
  Use Rapid Random Restart to find clauses
  After each generation of a new clause c
    Find the recall bin_r for c on the trainset
    If the *precision* $\times$ *recall* of c is best yet
      Replace c in $bin_{r,i}$

Determine Bin Threshold:
For each bin_j
  Find highest precision theory m and $L_m \in [1, K]$
  on trainset such that
    recall of "At least L of K clauses match
    examples" $\approx$ recall for bin_j

Evaluate On Testset:
Find precision and recall of testset using each
bin's "at least L of K" decision process

object and subject relations about the sentence grammar, as well as predicted parts of speech for each word in a relationship, for a total of 27 possible relations. For example, in the sentence "ykuD was transcribed by SigK RNA polymerase from T4 of sporulation," Link Parser reports that the noun 'yukD' is the subject of the verb 'transcribed', 'polymerase' and 'T4' are complements of 'transcribed', and 'RNA' and 'SigK' are modifiers of 'polymerase'.

We chose to ignore the lemma information, since we previously incorporated the stem of each word, and only focused on the 27 syntactic information predicates. We compare the inclusion versus exclusion of this enriched background information in our results.

3. Gleaner

Gleaner (Goadrich et al., 2004) is a randomized search method which collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an "at least N of these M clauses" thresholding method to combine sets of selected clauses. Pseudo-code for our algorithm appears in Table 2.

Gleaner uses Aleph (Srinivasan, 2003) as its underlying engine for generating clauses. As input, Aleph takes

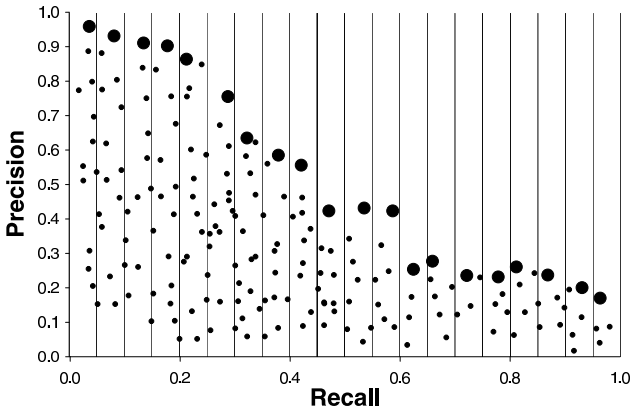


Figure 2. A sample run of Gleaner for one seed and 20 bins, showing each considered clause as a small circle, and the chosen clause per bin as a large circle. This is repeated for 100 seeds to gather 2,000 clauses (assuming a clause is found that falls into each bin for each seed).

background information in the form of either intensional or extensional predicates, a list of modes declaring how these predicates can be chained together, and a designation of one predicate as the "head" predicate to be learned. At a high-level overview, Aleph sequentially generates clauses for the positive examples by picking a random example to be a *seed*. This example is then saturated to create the *bottom clause*, i.e. every relation in the background knowledge that can be connected by relations to this example in a fixed number of steps. The bottom clause determines the possible search space for clauses. Aleph heuristically searches through the space of possible clauses until the "best" clause is found or time runs out. When enough clauses are learned to cover (almost) all of the positive training examples, the learned clauses are combined to form a theory. In our experiments, we will compare Gleaner to standard Aleph theories.

After initialization, the first stage of Gleaner learns a wide spectrum of clauses, as illustrated in Figure 2. We search for clauses using 100 random seed examples to encourage diversity. In our experiments, the recall dimension is uniformly divided into 20 equal sized bins, $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1.00]$. For each seed, we consider up to 25,000 possible clauses using Rapid Random Restart (Železný et al., 2003). As these clauses are generated, we compute the recall of each clause and determine into which bin the clause falls. Each bin keeps track of the best clause appearing in its bin for the current seed. We use the heuristic function *precision* $\times$ *recall* to determine the best clause. At the end of this search process, there will be 20 clauses collected for each seed and 100 seed examples for a total of 2,000 clauses (assuming a clause is found that falls

into each bin for each seed).

The second stage (modified slighly from (Goadrich et al., 2004)) takes place once all our clauses have been gathered using random search. Gleaner combines the clauses in each bin to create one large thresholded disjunctive clause per bin, of the form "At least L of these K clauses must cover an example in order to classify it as a positive." Each of these theories could generate their own recall-precision curves, by exploring all possible values for L on the tuneset, starting with $L = K$ and incrementally lowering the threshold to increase recall. These 20 curves will overlap in their recall and precision results, and we choose the theory which created the highest points along this combined curve on the tuneset, irrespective of the bin which generated the points. We will end up with 20 recall-precision points, one for each bin, that span the recall-precision curve.

A unique aspect of Gleaner is that each point in the recall-precision curve could be generated by a separate theory, instead of the usual setup to create a curve, where one hypothesis is transformed into many by ranking the examples and then finding different thresholds of classification. This separate-theory method is related to using the ROC convex hull created from separate classifiers (Fawcett, 2003). We believe using separate theories is a strength of our Gleaner approach, such that each theory, and therefore each point on our curves, is not hindered by the mistakes of previous points; each theory is totally independent of the others.

An end-user of Gleaner will be able to choose their preferred operating point from this recall-precision curve. Our algorithm will then be used to generate testset classifications using the closest bin to their desired recall results by using our found threshold L.

4. Results

There were two dimensions on which to vary our training methods, learning on data containing co-references or on data without co-references, and including the provided linguistic information (enriched) or using only the basic data. Tables 3 and 4 show the results of Gleaner on the testset data for all four combinations, using the restriction that the same word cannot be both agent and target in a relation[4]. A sample clause learned by Aleph can be found in Table 5. This clause has focused on the common property that agents are before targets, agents are nouns with internal capital letters and are complements of nouns which complement verbs, while targets are in noun phrases without negatively correlated words in the training set.

[4]For our challenge-task submission, we used all 936 possible test examples. Using the non-identical restriction resulted in a small increase in our precision results.

Table 3. Results of Gleaner, Aleph theory, and baseline all-positive prediction on LLL challenge task without co-reference.

Alg	Enriched	F1	Recall	Precision
Gleaner	-	41.7	79.6	28.3
	√	25.1	79.6	14.9
Aleph 1K	-	50.0	62.9	40.6
	√	31.0	59.2	21.0
Aleph 25K	-	30.7	44.4	23.5
	√	26.1	42.5	18.8
All Pos	*N/A*	20.1	100.0	11.2

Table 4. Results of Gleaner, Aleph theory, and baseline all-positive prediction on LLL challenge task with co-reference.

Alg	Enriched	F1	Recall	Precision
Gleaner	-	17.7	79.3	10.0
	√	18.5	82.7	10.4
Aleph 1K	-	31.6	51.7	22.7
	√	19.3	37.9	13.0
Aleph 25K	-	19.9	20.6	19.3
	√	19.1	24.1	15.9
All Pos	*N/A*	12.5	100.0	6.7

We chose our preferred operating point by choosing the bin with the highest F1 measure on the tuning set; these were bin [0.55, 0.60] on the basic dataset without co-reference, [0.65, 0.70] on the enriched dataset without co-reference and bin [0.90, 0.95] on the dataset with co-reference. With the enriched data, similar recall points can still be achieved, however there is a marked decrease in precision for the without co-reference dataset. We plan to explore the use of the enriched data from Link Parser (Temperly et al., 1999) in our future work on this and other information-extraction datasets.

We also show a comparison of Gleaner to two other algorithms. First, we examine the results of a single Aleph theory learned for each training set combination. We restrict each clause learned to have a min-

Table 5. Sample Clause with 20% Recall and 94% Precision on Without Co-reference Training Set

```
agent_target(A,T,S) :-
        n(A),
        complement_of_N_N(G,A),
        complement_by_V_PASS_N(G,_),
        word_parent(A,F),
        phrase_contains_some_internal_cap_word(F,_,A),
        word_parent(T,E),
        phrase_contains_no_arg_halfX_word(E,arg2,_),
        isa_np_segment(E),
        target_arg1_before_target_arg2(A,T).
```

where the variable A is the agent, T is the target, S is the sentence, and '_' indicates variables that only appear once in the clause.

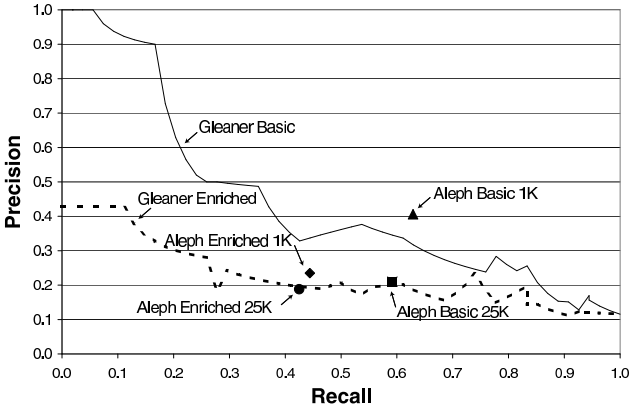


Figure 3. Recall-Precision Curves for Gleaner and Aleph on the dataset without co-reference.

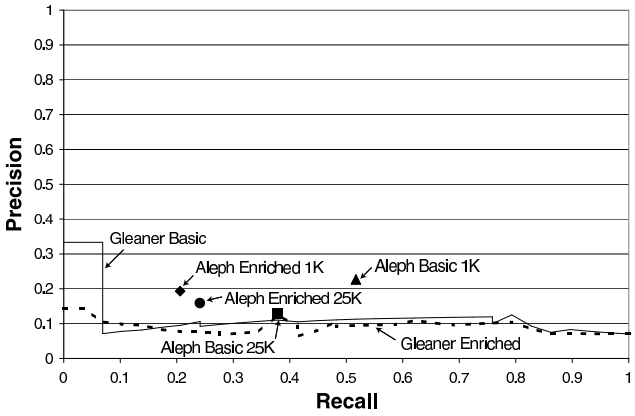


Figure 4. Recall-Precision Curves for Gleaner and Aleph on the dataset with co-reference.

imum precision of 75.0 and to cover a minimum of 5 positives in the training set. We also consider a maximum of both 1,000 and 25,000 clauses for each "best" clause in a theory. With the basic data, we see Aleph improves in precision, however recall is much lower that our results with Gleaner. We also notice a large drop in precision and recall between 1,000 clauses and 25,000 clauses, which we attribute to overfitting. Second, we compare to the algorithm of calling every example positive, which guarantees us 100% recall, and notice that Gleaner has an increase in precision over this baseline in both datasets.

Figure 3 shows recall-precision curves for Gleaner and recall-precision points for the Aleph theories on the dataset without co-reference, while Figure 4 shows results on the dataset with co-reference. Gleaner is able to span the whole recall-precision dimension, although with less than stellar results on the without co-reference dataset.. Gleaner seemed to suffer by not distinguishing well between the agent and target; when genic_interaction(A,B) was predicted, most often we also predicted genic_interaction(B,A), keeping the precision lower than 50%. Another cause of our low results could be the fact that sentences with genes and proteins but no relationships between them were not included in the training sets, but made up almost half of the testing set. This lack of negative sentences in the training sets hampered our ability to distinguish between good and bad sentences when learning clauses. Also, the size of the LLL challenge task was small in comparison to our previous work (Goadrich et al., 2004), creating the possibility of overfitting. Particularly affected were the enriched linguistic predicates and the statistical predicates, which focused on irrelevant words (e.g. specific gene and protein words like "sigma A" and "gerE"). Although collecting labeled

data for biomedical information extraction can be expensive, we believe the benefits are worth the cost.

5. Conclusions

This paper has explored two Inductive Logic Programming approaches to biomedical information extraction: Aleph, which learns many high-precision clauses that cover the training set, and Gleaner, which learns clauses from a wide spectrum of recall points and combines them to create broad thresholded theories. We developed a large number of background knowledge predicates which try to capture both the structure and semantics of biomedical text, and we evaluated these two algorithms on the Learning Language in Logic 2005 Challenge Task.

We believe there is much work remaining in the combination of ILP and biomedical information extraction. The logical structure of sentence parses as well as the biological semantic class information can be readily included in an ILP approach. This genic-interaction dataset was particularly interesting since neither the agent entity nor the target entity was a closed set, and there could be crossover between them. Also worth noting was the difference between the training set and testing set with respect to negative examples. We plan to further explore the issues which arose from using this dataset and perform cross-validation experiments to test for statistical significance of our results and to include negative sentences in the training set.

6. Acknowledgements

We gratefully acknowledge the funding from USA NLM Grant 5T15LM007359-02, USA NLM Grant 1R01LM07050-01, USA DARPA Grant F30602-01-2-0571, and USA Air Force Grant F30602-01-2-0571. Thanks to Burr Settles for help with parsing and tagging the sentences.

References

Blaschke, C., Hirschman, L., & Valencia, A. (2002). Information Extraction in Molecular Biology. *Briefings in Bioinformatics*, *3*, 154–165.

Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*.

Bunescu, R., Ge, R., Kate, R., Marcotte, E., Mooney, R., Ramani, A., & Wong, Y. (2004). Comparative Experiments on Learning Information Extractors for Proteins and their Interactions. *Journal of Artificial Intelligence in Medicine*, 139–155.

Eliassi-Rad, T., & Shavlik, J. (2001). A Theory-Refinement Approach to Information Extraction. *Proceedings of the 18th International Conference on Machine Learning*.

Fawcett, T. (2003). *ROC Graphs: Notes and Practical Considerations for Researchers* (Technical Report). HP Labs HPL-2003-4.

Goadrich, M., Oliphant, L., & Shavlik, J. (2004). Learning Ensembles of First-Order Clauses for Recall-Precision Curves: A Case Study in Biomedical Information Extraction. *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP)*. Porto, Portugal.

Kim, J.-D., Ohta, T., Teteisi, Y., & Tsujii, J. (2003). GENIA corpus - a semantically annotated corpus for bio-textmining. *Bioinformatics*, *19*.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning* (pp. 282–289). Morgan Kaufmann, San Francisco, CA.

Porter, M. (1980). An Algorithm for Suffix Stripping. *Program*, *14*, 130–137.

Ray, S., & Craven, M. (2001). Representing Sentence Structure in Hidden Markov Models for Information Extraction. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*.

Sang, E. F. T. K. (2001). Transforming a Chunker into a Parser. *Linguistics in the Netherlands*.

Shatkay, H., & Feldman, R. (2003). Mining the Biomedical Literature in the Genomic Era: An Overview. *Journal of Computational Biology*, *10*, 821–55.

Srinivasan, A. (2003). The Aleph Manual Version 4. *http://web.comlab.ox.ac.uk/ oucl/ research/ areas/ machlearn/ Aleph/*.

Temperly, D., Sleator, D., & Lafferty, J. (1999). An introduction to the Link Grammar Parser. *http://www.link.cs.wisc.edu/link/*.

Železný, F., Srinivasan, A., & Page, D. (2003). Lattice-Search Runtime Distributions may be Heavy-Tailed. *Proceedings of the 12th International Conference on Inductive Logic Programming 2002* (pp. 333–345). Springer Verlag.